### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertica® 9.0.x Documentation</td>
<td>4</td>
</tr>
<tr>
<td>Vertica 9.0.x Supported Platforms</td>
<td>11</td>
</tr>
<tr>
<td>Vertica 9.0.x New Features</td>
<td>37</td>
</tr>
<tr>
<td>Vertica Concepts</td>
<td>59</td>
</tr>
<tr>
<td>Installing Vertica</td>
<td>107</td>
</tr>
<tr>
<td>Getting Started</td>
<td>269</td>
</tr>
<tr>
<td>Administrator’s Guide</td>
<td>335</td>
</tr>
<tr>
<td>Analyzing Data</td>
<td>1423</td>
</tr>
<tr>
<td>Using Flex Tables</td>
<td>1729</td>
</tr>
<tr>
<td>Using Management Console</td>
<td>1905</td>
</tr>
<tr>
<td>SQL Reference Manual</td>
<td>2099</td>
</tr>
<tr>
<td>Security and Authentication</td>
<td>3993</td>
</tr>
<tr>
<td>Extending Vertica</td>
<td>4111</td>
</tr>
<tr>
<td>Connecting to Vertica</td>
<td>4377</td>
</tr>
<tr>
<td>Using Vertica on the Cloud</td>
<td>4919</td>
</tr>
<tr>
<td>Integrating with Apache Hadoop</td>
<td>4987</td>
</tr>
<tr>
<td>Integrating with Apache Kafka</td>
<td>5093</td>
</tr>
<tr>
<td>Integrating with Apache Spark</td>
<td>5191</td>
</tr>
<tr>
<td>Vertica Pulse</td>
<td>5217</td>
</tr>
<tr>
<td>Vertica Plug-In for Informatica</td>
<td>5325</td>
</tr>
<tr>
<td>Vertica Error Messages</td>
<td>5371</td>
</tr>
<tr>
<td>Glossary</td>
<td>5551</td>
</tr>
<tr>
<td>Third-Party Software Acknowledgements</td>
<td>5669</td>
</tr>
</tbody>
</table>
Legal Notices

Warranty
The only warranties for Open Text Corporation products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. OpenText shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted Rights Legend
Confidential computer software. Valid license from OpenText required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor’s standard commercial license.

Copyright Notice
© Copyright 2006 - 2023 Open Text Corporation

Trademark Notices
Adobe™ is a trademark of Adobe Systems Incorporated.
Apache® Hadoop® and Hadoop are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.
Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.
UNIX® is a registered trademark of The Open Group.
This product includes an interface of the ‘zlib’ general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.
Welcome to the Vertica 9.0.x online documentation. Published on 1/28/2023 at 1:56 AM. This PDF contains most of the content available on the My Vertica portal except for the API documentation. See the Vertica documentation page for the latest documentation.
Release Notes

You can find the latest Vertica 9.0.x Release Notes at http://my.vertica.com/docs.

You are viewing downloaded documentation. Vertica regularly updates the product documentation posted on the My Vertica portal. See https://vertica.com/documentation/vertica/ for the latest documentation.
Other Resources

- **Vertica Software Downloads**

- **Vertica User Community**: Technical resources for customers: Blog, forum, knowledge base, partner integration guides, troubleshooting checklists, and videos

- **The QuickStart BI Examples** page has technology samples powered by Vertica. Each QuickStart includes documentation and an introductory video.

- **Resources for Planning for and Configuring Hardware** explains how to configure hardware for Vertica.

- **Syntax Conventions** for Vertica documentation
Vendor Information

55 Blue Sky Drive, Suite 102
Burlington, MA 01803
Phone: +1 617 386 4400
E-Mail: contactvertica@microfocus.com
Web site: http://www.vertica.com
# Syntax Conventions

These are the syntax conventions used throughout the Vertica documentation.

<table>
<thead>
<tr>
<th>Syntax Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text without brackets or braces</td>
<td>Content that you type as shown, such as <code>CREATE TABLE</code>.</td>
</tr>
<tr>
<td><code>table name</code></td>
<td><strong>Italics</strong>: Descriptive placeholder text that you replace with an appropriate identifier or expression.</td>
</tr>
<tr>
<td><code>{ param1, param2 }</code></td>
<td><strong>Curly braces</strong>: Required parameters or arguments. For example: `LIGHT_STATUS { ON</td>
</tr>
<tr>
<td>`{ this</td>
<td>that }`</td>
</tr>
<tr>
<td><code>[param1]</code></td>
<td><strong>Square brackets</strong>: Optional statement items. For example, <code>CREATE TABLE [schema_name.]table_name</code> indicates that <code>schema_name</code> is optional.</td>
</tr>
<tr>
<td><code>column_def[,...]</code></td>
<td><strong>Square brackets with comma and ellipses (...)</strong>: A parameter option that you can repeat. For example, <code>column_def[,...]</code> means you can specify one or more comma-separated <code>column_def</code> options.</td>
</tr>
<tr>
<td><code>start_json_display</code></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /> <strong>Vertical ellipses</strong>: Indicate text intentionally omitted for readability, such as in an example of multi-row result sets.</td>
</tr>
<tr>
<td><code>end_display</code></td>
<td><img src="https://via.placeholder.com/150" alt="Image" /> <strong>Indentation</strong>: Indented text: Formatting to maximize readability for</td>
</tr>
<tr>
<td>Syntax Convention</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>syntax sub-options. Indentation is never required, since SQL is a free-form language.</td>
</tr>
</tbody>
</table>
Vertica 9.0.x Supported Platforms

Welcome to Vertica Analytics Platform Supported Platforms. This document details platform support for the various components of Vertica 9.0.x.
Vertica Server and Vertica Management Console

Supported Operating Systems and Operating System Versions

[[[Undefined variable _Branding_Variables._Company_Acronym]]] supports Vertica Analytic Database 9.0.x running on the following 64-bit operating systems and versions on x86_64 architecture.

Important: This document reflects what has been tested with Vertica Analytic Database 9.0.x. Be aware that operating system vendors and communities release updates and patches for all versions of their operating systems on their own schedules. Such updates to these operating systems may or may not coincide with the release schedule for Vertica. While other versions of the operating systems listed may have been successfully deployed by customers in their environments, stability and performance of these configurations may vary. If you choose to run Vertica on an operating system version not listed in this document and experience an issue, the Vertica Support team may ask you to reproduce the issue using one of the configurations described in this document to aid in troubleshooting. Depending on the details of the case, the Support team may also ask you to enter a support ticket with your operating system vendor.

When there are multiple minor versions supported for a major operating system release, [[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you run Vertica on the latest minor version listed in the supported versions list. For example, if you run Vertica on a Red Hat Enterprise Linux 7.x release, [[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends you upgrade to or be running the latest supported RHEL 7.x release, which is 7.4.

Red Hat Enterprise Linux

- Versions 6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4
Important: You cannot perform an in-place upgrade of your current Vertica Analytic Database from Red Hat Enterprise Linux 6.6–6.8 to Red Hat Enterprise Linux 7.0/7.3/7.4. For information on how to upgrade to Red Hat Enterprise Linux 7, see the Upgrading Your Operating System on Nodes in Your Vertica Cluster. For information on changes to the operating system for Red Hat Enterprise Linux 7, see the Red Hat Enterprise Linux 7 documentation.

CentOS

- Versions 6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4

Important: You cannot perform an in-place upgrade of your current Vertica Analytic Database from CentOS 6.6–6.8 to CentOS 7.0/7.3/7.4. For information on how to upgrade to CentOS 7.0/7.3/7.4, see the Upgrading Your Operating System on Nodes in Your Vertica Cluster.

SUSE Linux Enterprise Server

- Version 12 SP3*
- Version 12 SP2
- Version 11.0 SP3**

* Vertica supports SUSE 12 SP3 on Vertica 9.0.1-3 and later.
** Vertica is phasing out support for Vertica on this operating system. See End-of-Support and Deprecation Notices for more information.

Note: SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) supports the ext3 file system.

Amazon Linux

Vertica supports:

- Amazon Linux 2017.09
Important:

- You must use RHEL/CentOS rpms for Amazon Linux 2017.09.
- For Amazon Linux only, Vertica supports i3.2xlarge and i3.4xlarge instances, in addition to the standard supported instances. See Supported Instance Types.
- There is no Vertica AMI based on Amazon Linux.
- You can only deploy Amazon Linux 2017.09 in AWS.

Oracle Enterprise Linux

Vertica supports the following Oracle Enterprise Linux (OEL) versions. Vertica only supports Red Hat Compatible Kernels. Vertica is not supported on unbreakable kernels (kernels with a uel suffix).

- Versions 6.7, 6.8, 6.9, 7.3, 7.4

Debian Linux

- Versions 7.6, 7.7, 8.5, 8.9

Ubuntu

- Version 12.04 LTS*, 14.04 LTS, 16.04 LTS**

* Vertica is phasing out support on this operating system. See End-of-Support and Deprecation Notices for more information.

** Vertica 9.0.1-9 and later 9.0.1 hotfixes support Ubuntu 16.04 LTS.

Supported File Systems

Vertica Analytic Database Enterprise Edition has been tested on all supported Linux platforms running the ext4 file system. For the Vertica Analytic Database I/O profile, the ext4 file system is considerably faster than ext3.
While other file systems have been successfully deployed by some customers, Vertica Analytic Database cannot guarantee performance or stability of the product on these file systems. In certain support situations, you may be asked to migrate off of these unsupported file systems to help you troubleshoot or fix an issue. In particular, several file corruption issues have been linked to the use of XFS with Vertica; [[[Undefined variable _Branding_Variables._Company_Acronym]]] strongly recommends not using it in production.

Vertica Analytic Database supports Linux Volume Manager (LVM) on all supported operating systems. Your LVM version must be 2.02.66 or later, and must include device-mapper version 1.02.48 or later. For information on requirements and restrictions, see the section, Vertica Support for LVM.

**Supported Browsers for Vertica Management Console**

Vertica Analytic Database 9.0.x Management Console is supported on the following web browsers:

- Internet Explorer 10 and later
- Firefox 31 and later
- Google Chrome 38 and later

**Vertica Server and Management Console Compatibility**

Management Console (MC) 9.0.x is compatible with the latest hotfix version of Vertica server 7.2.3 and above.
Vertica 9.0.x Client Drivers

Vertica provides JDBC, ODBC, OLE DB, Python, vsql, and ADO.NET client drivers. Download the latest drivers from: Vertica Client Drivers. Choose from these drivers:

- Linux and UNIX-like platforms: ODBC, JDBC, Python, and vsql clients. See Installing the Client Drivers on Linux and UNIX-Like Platforms.


- Mac OS X platforms: ODBC and vsql clients. See Installing the Client Drivers on Mac OS X.

- The cross-platform JDBC client .jar file available for installation on all platforms.

To view a list of driver and server version compatibility, see Vertica Analytic Database Driver/Server Compatibility.

ADO.NET and OLE DB Drivers

The ADO.NET and OLE DB drivers are supported on the following platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Processor</th>
<th>Supported Versions</th>
<th>.NET Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>x86 (32-bit)</td>
<td>Windows 7, Windows 8, Windows 10</td>
<td>Microsoft .NET Framework 3.5 SP1 or later</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>x64 (64-bit)</td>
<td>Windows 7, Windows 8, Windows 10</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows Server</td>
<td>x64 (64-bit)</td>
<td>2012, 2012 R2, 2016</td>
<td></td>
</tr>
</tbody>
</table>
**JDBC Driver**

All non-FIPS JDBC drivers are supported on any Java 5-compliant platform. (Java 5 is the minimum.)

The Vertica 9.0. FIPS-enabled JDBC driver requires RHEL 6.6.

**Python Driver**

The Python driver requires Python version 2.7.x, and is supported on the following platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Processor</th>
<th>Supported Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>x86_64</td>
<td>6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4</td>
</tr>
<tr>
<td>CentOS</td>
<td>x86_64</td>
<td>6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4</td>
</tr>
<tr>
<td>SUSE Linux Enterprise</td>
<td>x86_64</td>
<td>11.0 SP3*, 12 SP2, 12 SP3**</td>
</tr>
<tr>
<td>Oracle Enterprise Linux (Red Hat Compatible Kernel only)</td>
<td>x86_64</td>
<td>6.7, 6.8, 6.9, 7.3, 7.4</td>
</tr>
<tr>
<td>Debian Linux</td>
<td>x86_64</td>
<td>7.6, 7.7, 8.5, 8.9</td>
</tr>
<tr>
<td>Solaris*</td>
<td>x86_64</td>
<td>10</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>x86_64</td>
<td>12.04LTS*, 14.04 LTS, 16.04 LTS</td>
</tr>
<tr>
<td>Amazon Linux</td>
<td>x86_64</td>
<td>2017.09</td>
</tr>
</tbody>
</table>

* Vertica is phasing out support on this operating system. See End-of-Support and Deprecation Notices for more information.

** Vertica supports SUSE 12 SP3 on Vertica 9.0.1-3 and later.
# ODBC Driver

Vertica Analytic Database provides both 32-bit and 64-bit ODBC drivers. Vertica 9.0.x ODBC drivers are supported on the following platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Processor</th>
<th>Supported Versions</th>
<th>Driver Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>x86 (32-bit)</td>
<td>Windows 7, Windows 8, Windows 10</td>
<td>Microsoft ODBC MDAC 2.8</td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td>x64 (64-bit)</td>
<td>Windows 7, Windows 8, Windows 10</td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows Server</td>
<td>x64 (64-bit)</td>
<td>2012, 2012 R2, 2016</td>
<td></td>
</tr>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>x86_64</td>
<td>6.6, 6.7, 6.8, 6.9, 6.9, 7.0, 7.3, 7.4</td>
<td>iODBC 3.52.6 and higher</td>
</tr>
<tr>
<td>FIPS-compliant Red Hat Enterprise Linux</td>
<td>x86_64</td>
<td>6.6</td>
<td>unixODBC 2.3.0 and higher</td>
</tr>
<tr>
<td>CentOS</td>
<td>x86_64</td>
<td>6.6, 6.7, 6.8, 7.0, 7.3, 7.4</td>
<td>DataDirect 5.3 and 6.1 and higher</td>
</tr>
<tr>
<td>SUSE Linux Enterprise</td>
<td>x86_64</td>
<td>11.0 SP3*, 12 SP2, 12 SP3**</td>
<td></td>
</tr>
<tr>
<td>Oracle Enterprise Linux (Red Hat Compatible Kernel only)</td>
<td>x86_64</td>
<td>6.7, 6.8, 6.9, 7.3, 7.4</td>
<td></td>
</tr>
<tr>
<td>Ubuntu</td>
<td>x86_64</td>
<td>12.04LTS*, 14.04 LTS, 16.04 LTS</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td>Processor</td>
<td>Supported Versions</td>
<td>Driver Manager</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>AIX</td>
<td>PowerPC-64</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td>IA-64</td>
<td>11i V3**</td>
<td></td>
</tr>
<tr>
<td>Debian Linux</td>
<td>x86_64</td>
<td>7.6, 7.7, 8.5, 8.9</td>
<td></td>
</tr>
<tr>
<td>Solaris*</td>
<td>SPARC-64</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Mac OS X</td>
<td>x86_64</td>
<td>10.10, 10.11, 10.12</td>
<td></td>
</tr>
<tr>
<td>Amazon Linux</td>
<td>x86_64</td>
<td>2017.09</td>
<td></td>
</tr>
</tbody>
</table>

* Vertica is phasing out support on this operating system. See [End-of-Support and Deprecation Notices](#) for more information.

** Vertica supports SUSE 12 SP3 on Vertica 9.0.1-3 and later.
**Vertica Analytic Database Driver/Server Compatibility**

This section provides information on compatibility for the Vertica Analytic Database driver and server versions.

**Note:** SHA password security is supported on client driver and server versions 7.1.x and later.

The following table indicates that, in general, all clients are forward compatible.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Clients</td>
<td>6.1.x</td>
<td>6.1.x, 7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>7.0.x</td>
<td></td>
<td>7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>7.1.x</td>
<td></td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>7.2.x</td>
<td></td>
<td>7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>8.0.x</td>
<td></td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>8.1.x</td>
<td></td>
<td>8.1.x, 9.0.x</td>
</tr>
<tr>
<td>9.0.x</td>
<td></td>
<td>9.0.x</td>
</tr>
</tbody>
</table>

The following table lists FIPS 140-2 compatible clients.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIPS-enabled ODBC</td>
<td>8.0.x</td>
<td>8.0.x, 8.1.x</td>
</tr>
<tr>
<td>FIPS-enabled ODBC</td>
<td>8.1.x</td>
<td>8.0.x, 8.1.x</td>
</tr>
<tr>
<td>FIPS-enabled ODBC</td>
<td>9.0.x</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>FIPS-enabled JDBC</td>
<td>8.1.x</td>
<td>8.1.x</td>
</tr>
<tr>
<td>FIPS-enabled JDBC</td>
<td>9.0.x</td>
<td>8.1.x, 9.0.x</td>
</tr>
</tbody>
</table>

The following table indicates that the ODBC client is backward compatible.
The following table indicates that the JDBC and ADO.NET clients are backward compatible.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC (backwards compatibility)</td>
<td>8.1.x</td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x</td>
</tr>
<tr>
<td>ODBC (backwards compatibility)</td>
<td>9.0.x</td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
</tbody>
</table>

**Note:** Vertica Release 9.0.x and later adds backwards compatibility for the ODBC client driver. The 9.0.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see [Data Source Name (DSN) Connection Properties](#).

---

**vsqI Client**

The Vertica vsqI client is included in all client packages; it is not available for download separately. The vsqI client is supported on the following platforms:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>x86, x64</td>
</tr>
<tr>
<td>- Windows 2012, 2012 R2, 2016, all variants</td>
<td></td>
</tr>
<tr>
<td>- Windows 7, all variants</td>
<td></td>
</tr>
<tr>
<td>- Windows 8, all variants</td>
<td></td>
</tr>
<tr>
<td>- Windows 10</td>
<td></td>
</tr>
</tbody>
</table>

---
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux 6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4</td>
<td>x86, x64</td>
</tr>
<tr>
<td>CentOS 6.6, 6.7, 6.8, 6.9, 7.0, 7.3, 7.4</td>
<td>x86, x64</td>
</tr>
<tr>
<td>FIPS-compliant Red Hat Enterprise Linux 6.6</td>
<td>x64</td>
</tr>
<tr>
<td>SUSE Linux Enterprise 11.0 SP3*, 12 SP2, 12 SP3**</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Oracle Enterprise Linux 6.7, 6.8, 6.9, 7.3, 7.4 (Red Hat Compatible Kernels only)</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Ubuntu 12.04LTS*, 14.04 LTS, 16.04 LTS</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Debian Linux 7.6, 7.7, 8.5, 8.9</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Solaris 10*</td>
<td>SPARC-64</td>
</tr>
<tr>
<td>AIX 7.1</td>
<td>PowerPC-64</td>
</tr>
<tr>
<td>HP-UX 11i V3</td>
<td>IA64</td>
</tr>
<tr>
<td>Mac OS X 10.10, 10.11, 10.12</td>
<td>x86, x64</td>
</tr>
<tr>
<td>Amazon Linux 2017.09</td>
<td>x86, x64</td>
</tr>
</tbody>
</table>

* Vertica is phasing out support on this operating system. See End-of-Support and Deprecation Notices for more information.

** Vertica supports SUSE 12 SP3 on Vertica 9.0.1-3 and later.
Perl and Python Requirements

You can use Vertica's ODBC driver to connect applications written in Perl or Python to the Vertica Analytic Database.

Perl

To use Perl with Vertica, you must install the Perl driver modules (DBI and DBD::ODBC) and a Vertica ODBC driver on the machine where Perl is installed. The following table lists the Perl versions supported with Vertica 9.0.x.

<table>
<thead>
<tr>
<th>Perl Version</th>
<th>Perl Driver Modules</th>
<th>ODBC Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>DBI driver version 1.609</td>
<td>See Vertica 9.0.x Client Drivers.</td>
</tr>
<tr>
<td>5.10</td>
<td>DBD::ODBC version 1.22</td>
<td></td>
</tr>
</tbody>
</table>

Python

To use Python with Vertica, you must install the Vertica Python Client or the pyodbc module and a Vertica ODBC driver on the machine where Python is installed. The following table lists the Python versions supported with Vertica 9.0.x:

<table>
<thead>
<tr>
<th>Python Version</th>
<th>Python Driver Module</th>
<th>ODBC Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.6</td>
<td>pyodbc 2.1.6</td>
<td>See Vertica 9.0.x Client Drivers.</td>
</tr>
<tr>
<td>2.7.x</td>
<td>Vertica Python Client (Linux only)</td>
<td></td>
</tr>
<tr>
<td>2.7.3</td>
<td>pyodbc 3.0.6</td>
<td></td>
</tr>
<tr>
<td>3.3.4</td>
<td>pyodbc 3.0.7</td>
<td></td>
</tr>
</tbody>
</table>
Vertica SDKs

This section details software requirements for running User Defined Extensions (UDxs) developed using the Vertica SDKs.

C++ SDK

The Vertica cluster does not have any special requirements for running UDxs written in C++.

Java SDK

Your Vertica cluster must have a Java runtime installed to run UDxs developed using the Vertica Java SDK. Vertica has tested the following Java Runtime Environments (JREs) with this version of the Vertica Java SDK:

- Oracle Java Platform Standard Edition 6 (version number 1.6)
- Oracle Java Platform Standard Edition 7 (version number 1.7)
- Oracle Java Platform Standard Edition 8 (version number 1.8)
- OpenJDK 6 (version number 1.6)
- OpenJDK 7 (version number 1.7)
- OpenJDK 8 (version number 1.8)

Python SDK

The Vertica Python SDK does not require any additional configuration or header files.
R Language Pack

The Vertica R Language Pack provides version 3.2.5 of the R runtime and associated libraries for interfacing with Vertica. You install the R Language Pack on the Vertica server.
FIPS 140-2 Supported Platforms

Vertica uses a certified OpenSSL FIPS 140-2 cryptographic module to meet the security standards set by the National Institute of Standards and Technology (NIST) for Federal Agencies in the United States or other countries.

FIPS-enabled Vertica requires the following:

- Red Hat Linux 6.6, a FIPS compliant operating system
- OpenSSL 1.0.1e
- A user-generated certificate signed by an approved Certificate Authority
- TLS 1.2 to support the server-client connection for a FIPS-enabled system

Supported Drivers

Vertica supports the following client drivers for FIPS-compliance:

- vsql
- ODBC
- JDBC

Important: FIPS-enablement is not supported in the Management Console.

For more information see Federal Information Processing Standard.
Vertica Integrations for Hadoop

OpenText has tested Vertica 9.0.x with the following Hadoop distributions. OpenText expects Vertica to work with subsequent Hadoop distributions, and tests these later distributions as soon as practical.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Versions Tested with Vertica Server 9.0.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloudera (CDH)</td>
<td>• 5.7</td>
</tr>
<tr>
<td></td>
<td>• 5.8</td>
</tr>
<tr>
<td></td>
<td>• 5.11</td>
</tr>
<tr>
<td>HortonWorks Data Platform (HDP)</td>
<td>• 2.4</td>
</tr>
<tr>
<td></td>
<td>• 2.5</td>
</tr>
<tr>
<td></td>
<td>• 2.6</td>
</tr>
<tr>
<td>MapR</td>
<td>• 5.1</td>
</tr>
<tr>
<td></td>
<td>• 5.2</td>
</tr>
</tbody>
</table>

You must apply patches for the following issues: HDFS-8855, HDFS-8696, HDFS-7280, HDFS-7279, HDFS-7270, and HDFS-7945. See your Hadoop vendor documentation for further instructions.
Packs, Plug-Ins, and Connectors for Partner Products

[[[Undefined variable _Branding_Variables._Company_Acronym]]] provides the following optional module for Vertica client machines.

**Informatica PowerCenter Plug-In**

The Vertica plug-in for Informatica PowerCenter is supported on the following platforms:

<table>
<thead>
<tr>
<th>Plug-in Version</th>
<th>Informatica PowerCenter Versions</th>
<th>Vertica Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>9.x</td>
<td>6.x (limited functionality)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.x (all enhancements)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.x (all enhancements)</td>
</tr>
</tbody>
</table>

For a complete list of supported client drivers, see Vertica 9.0.x Client Drivers.

For more information about the Informatica PowerCenter Plug-in, see Introduction to Using Informatica PowerCenter with Vertica.
Vertica on Amazon Web Services

Vertica provides a preconfigured AMI for users who want to run Vertica Analytic Database on Amazon Web Services (AWS). This AMI allows users to configure their own storage and has been configured for and tested on AWS. This AMI is the officially supported version of Vertica Analytic Database for AWS.

Note that Vertica develops AMIs on a slightly different schedule than our product release schedule. Therefore, AMIs will be available for Vertica releases sometime following the initial release of Vertica software.
Vertica in a Virtualized Environment

Vertica runs in the following virtualization environment:

Important: Vertica does not support suspending a virtual machine while Vertica is running on it.

Host

- VMware virtual machine hardware version 10:
  - ESXi 5.5
  - Fusion 6.x
  - Workstation 10.x
  - Player 6.x
- The number of virtual machines per host did not exceed the number of physical processors
- CPU frequency scaling turned off at the host level and for each virtual machine
- VMware parameters for hugepages set at version 5.5 defaults

Input/Output

- Measured by vioperf concurrently on all Vertica nodes. When running vioperf, provide the –duration=2min option and start on all nodes concurrently
- 25 megabytes per second per core of write
- 20+20 megabytes per second per core of rewrite
- 40 megabytes per second per core of read
- 150 seeks per second of latency (SkipRead)
- Thick provisioned disk, or pass-through-storage
Network

- Dedicated 10G NIC for each Virtual Machine
- No oversubscription at the switch layer, verified with vnetperf

Processor

- Architecture of Sandy Bridge (HP Gen8 or higher)
- 8 or more virtual cores per virtual machine
- No oversubscription
- vcpuperf time of no more than 12 seconds (≈ 2.2 GHz clock speed)

Memory

- Pre-allocate and reserve memory for the VM
- 4G per virtual core of the virtual machines

Vertica has tested the configuration above. While other virtualization configurations may have been successfully deployed by customers in development environments, performance of these configurations may vary. If you choose to run Vertica on a different virtualization configuration and experience an issue, the Vertica Support team may ask you to reproduce the issue using the configuration described above, or in a bare-metal environment, to aid in troubleshooting. Depending on the details of the case, the Support team may also ask you to enter a support ticket with your virtualization vendor.
Vertica Integration for Apache Kafka

You can use Vertica with the Apache Kafka message broker. For more information on Kafka integration, refer to How Vertica and Apache Kafka Work Together.

Kafka Versions

Vertica supports the following Kafka distributions:

<table>
<thead>
<tr>
<th>Apache Kafka Versions</th>
<th>Vertica Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>7.2 SP2 and later</td>
</tr>
<tr>
<td>0.10</td>
<td>8.1 and later</td>
</tr>
</tbody>
</table>

Java Versions

The data streaming job scheduler needs to make a JDBC connection to the target database, and requires Java 7.0 or later to run.
Vertica Support for LVM

Vertica Analytic Database 9.0.x supports Linux Volume Manager (LVM) on all supported operating systems.

LVM Version Supported

Vertica Analytic Database supports LVM version 2.02.66 or later, and must include device-mapper version 1.02.48 or later.

LVM Configuration Notes

In configuring LVM:

- When you create logical volumes with the `lvcreate` command, use the `readahead` option to set the read ahead sector count to greater than 2048 KB.
- You can use the default settings for all other LVM options.

LVM Restrictions

The following limitations apply to LVM support:

- Your logical volume file system type must be ext4.

  Note: SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) supports the ext3 file system. However, Vertica is phasing out support for this operating system. See End-of-Support and Deprecation Notices for more information.

- You cannot have physical drives shared across several nodes.
- Vertica supports linear logical volumes only. Vertica does not support striped or mirrored logical volumes.
- Vertica supports extending logical volumes (\texttt{lvextend}), but not reducing the size of a logical volume.

- Vertica recommends frequent backups.

- Vertica does not support LVM backup and restore, such as LVM snapshot and merge. Use the Vertica backup file vbr.py.

- Vertica does not support LVM space reclamation. Space reclamation is duplicated with reducing the size of a logical volume.

- Vertica does not support LVM migration. Use Vertica Copy operations.

- Vertica does not support LVM high availability. Use Vertica high availability capabilities.

- Vertica does not support LVM RAID. Configure RAID at the disk controller level.
Vertica Integration for Apache Spark

You can use the Vertica Connector for Apache Spark to transfer data between Vertica and Apache Spark. Vertica supports the following Apache Spark versions:

<table>
<thead>
<tr>
<th>Apache Spark Version</th>
<th>Scala Version</th>
<th>Vertica Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6*</td>
<td>2.10*</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>2.0</td>
<td>2.10*, 2.11</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>2.1</td>
<td>2.10*, 2.11</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>2.2</td>
<td>2.11</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
</tbody>
</table>

* Vertica is phasing out support for these products.

Note: Vertica 8.0.x is compatible with Spark 2.0 and 2.1 and Scala versions 2.10 and 2.11 when you use the Spark connector distributed with Vertica 8.1.x.

For more information on Apache Spark integration, refer to Integrating with Apache Spark.
End-of-Support and Deprecation Notices

These end-of-support notices apply to specific client, Linux, Hadoop, and Kafka distributions.

End-of-Support Notices

Vertica is phasing out support for certain client platforms and server distributions. See Deprecation Notices.

Deprecation Notices

Vertica is phasing out support for the following clients on the listed platforms.

<table>
<thead>
<tr>
<th>Client</th>
<th>End of Support for Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>All client drivers</td>
<td>SUSE 11 SP3</td>
</tr>
<tr>
<td>All client drivers</td>
<td>Solaris</td>
</tr>
</tbody>
</table>

For a list of supported platforms for Vertica clients, see Vertica 9.0.x Client Drivers.

Vertica is phasing out support for the following Linux distributions.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>End of Support for Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSE</td>
<td>11 SP3</td>
</tr>
<tr>
<td>Solaris</td>
<td>All</td>
</tr>
</tbody>
</table>

For a list of supported server operating systems, see Vertica Server and Vertica Management Console.

Effective 9.0.1, Vertica no longer provides support for SQL Server 2008.
Vertica 9.0.x New Features

Welcome to Vertica Analytics Platform New Features. This guide briefly describes the new features introduced in the most recent releases of Vertica and provides references to detailed information in the documentation set.

To see a list of known and fixed issues in the most current release, see the Vertica Release Notes, which you can find at https://my.vertica.com/docs/ReleaseNotes/9.0.x/Vertica_9.0.x_Release_Notes.htm.
In This Chapter

- Vertica 9.0. New Features and Changes
- Deprecated and Retired Functionality
Vertica 9.0.1 New Features and Changes

Read the topics in this section for information about new and changed functionality in Vertica 9.0.x.

In This Section

- Eon Mode Beta
- Management Console
- Loading Data
- Client Libraries
- Data Analysis
- Backup, Restore, and Recovery
- Apache Hadoop Integration
- Security and Authentication
- Spark Integration

Eon Mode Beta

Use the Administration Tools to Create an Eon Mode Beta Database

In Vertica 9.0.1, the Administration Tools, a graphical user interface that works in terminal windows, includes the option to create new Eon Mode Beta databases. Previously, this option
was available only through the command line and Management Console with Provisioning.

Eon Mode Beta Now Supports Adding and Deleting Columns

You can now use `ALTER TABLE` to add and delete table columns. Previously this functionality was available only in Enterprise Mode.

System Tables and Functions

The following have been added to Eon Mode Beta:

- **SYNC_CATALOG**: this function immediately synchronizes the catalog to shared storage to allow revive as of the current catalog version in the case of an imminent crash.
- **CATALOG_SYNC_STATE**: this system table indicates the last time a node synchronized its catalog.
- **CATALOG_TRUNCATION_STATUS**: this system table indicates how far behind a catalog version may be after you revive a cluster.

Management Console

Tag AWS Instances

To help you manage your instances, Management Console (MC) with Provisioning lets you tag instances when you create them with the Cluster Creation wizard.

You can define a key and an optional value when you tag. Tagging assigns metadata to the instance for easier resource management. For example, tags can indicate an instance's owner and purpose.

MC with Provisioning is available on the AWS marketplace.
Loading Data

Evaluation Errors Can Now Be Rejected

If errors occur during parsing, you can use the REJECTED DATA parameter to COPY to write rejected data to the file system or a table and continue loading data. Previously, errors during transformations instead aborted the load. You can now treat transformation errors the same way as parsing errors.

This feature is disabled by default because it can have performance implications. Rejecting rows during evaluation is more expensive than rejecting them during parsing, so a load with many expression errors could be much slower than a load of the same size without the errors. To enable rejection of expression errors, set the CopyFaultTolerantExpressions configuration parameter to 1. Consider using the REJECTMAX parameter to limit rejections and abort if too many are present.

For more information about this feature, see Transforming Data During Loads in the Administrator's Guide.

Client Libraries

Vertica version 9.0.0 introduced support for the UUID data type. Most 9.0.0 client libraries provided basic support for the new data type. With version 9.0.1, client libraries ADO.NET, ODBC, and OLE DB fully support the UUID data type.

For more information, see:

- Vertica and ODBC Data Type Translation
- ADO.NET Data Types
Data Analysis

Machine Learning for Predictive Analytics

Machine Learning for Predictive Analytics now supports the following functionality:

- Support for **Random Forest for Regression**.
- The ability to upgrade models from a previous version, using the **UPGRADE_MODEL** function.
- A new categorical summary function, named **SUMMARIZE_CATCOL**. This function provides a statistical summary of categorical data.
- Extended support for the **CROSS_VALIDATE** function to be applied to Naive Bayes.

Backup, Restore, and Recovery

VBR Wildcard Support

Vertica supports the use of wildcard characters to include or exclude database objects from your backup, restore, and replication tasks. You can use wildcards in your vbr.ini file or as vbr command line parameters. For more information, refer to Using Wildcards with Backup, Restore, and Replicate.

S3 Backup Encryption

Backups made to Amazon S3 can be encrypted using native server-side S3 encryption capability. For more information, refer to Encrypting Backups to S3.
Object Restore and Replication to a Newer Version of Vertica

Beginning with version 9.0.0-2, Vertica supports object replication and restore to a version up to one minor version later than the current database version. For example, you can replicate or restore objects from a version 9.0.0-2 database to a version 9.0.1 database.

Apache Hadoop Integration

HCatalog Connector Supports Ranger for Hive Authorization

The HCatalog Connector now integrates with Ranger to manage authorization for Hive data. (Support for a similar service, Sentry, was added previously.) You must connect to Hive using HiveServer2 (the default), not WebHCat, to use this feature.

For more information, see the HCatalog Connector section of Configuring Kerberos in Integrating with Apache Hadoop.

HCatalog Connector Supports HA Metastore

If HiveServer2 uses High Availability Metastore, you can direct the HCatalog Connector to take advantage of it. When using CREATE HCATALOG SCHEMA, you can specify a comma-delimited list for the value of the HOSTNAME parameter. Alternatively, you can omit the parameter and Vertica reads it from hive-site.xml.

Security and Authentication

System Table Access

System table access is now enhanced as follows:
You can now grant and revoke privileges on system tables to non-superusers with `GRANT (Table)` and `REVOKE (Table)`, respectively.

Two new Vertica functions, `RELEASE_SYSTEM_TABLES_ACCESS` and `RESTRICT_SYSTEM_TABLES_ACCESS`, respectively enable and restrict access to system tables for the current session.

Spark Integration

The Vertica Connector for Apache Spark now supports Spark 2.2.

The Vertica Spark Connector can now use the Parquet file format for intermediate data storage, as well as ORC format. See Intermediate Data File Format Settings for more information.
Vertica 9.0. New Features and Changes

Read the topics in this section for information about new and changed functionality in Vertica 9.0.x.

In This Section

- Upgrade and Installation
- Table Data Management
- Data Analysis
- Apache Hadoop Integration
- Loading Data
- Eon Mode Beta
- Management Console
- Security and Authentication
- SQL Functions and Statements
- Supported Data Types

Upgrade and Installation

Unique Model Names

As of this release, model names must be unique among all names of sequences, tables, projections, views, and models within the same schema. If the upgrade finds a model with the same name as another object in the same schema, it renames the model as follows:

```
model-name >> model-name_vN
```
where $N$ is an integer $\geq 0$. If $model\_name\_vN$ identifies an object that already exists in the schema, Vertica increments $N$ until it finds a name that is unique in the schema. After the upgrade is complete, check the Vertica log for all name changes.

**Model Schema Restrictions**

As of this release, you can create models only in public and user-created schemas (excluding HCatalog). Attempts to create models elsewhere return an error. If the upgrade finds a model that was created in a schema that does not conform with this restriction, it moves the model to the public schema. If another object of the same name already exists in public, the upgrade renames the moved model as follows:


Where $N$ is an integer $\geq 0$. If $model\_name\_vN$ identifies an object that already exists in the schema, Vertica increments $N$ until it finds a name that is unique in the schema. After the upgrade is complete, check the Vertica log for all changes of this type.

**Table Data Management**

This section summarizes new options and changes in Vertica Analytic Database 9.0, for managing table data in a Vertica database.

**Partition Grouping and Hierarchical Partitioning**

You can now consolidate partitions into groups that minimize use of ROS storage. Reducing the number of ROS containers to store partitioned data helps facilitate DML operations such as DELETE and UPDATE, and avoid ROS pushback. For example, you can group date partitions by year. By doing so, the Tuple Mover allocates ROS containers for each year group, and merges individual partitions into these ROS containers accordingly.

The new meta-function `CALCULATION_HIERARCHY_DAY` leverages partition grouping. This function organizes a table's date partitions into a hierarchy of groups: the oldest date partitions are grouped by year, more recent partitions are grouped by month, and the most recent date partitions remain ungrouped. Grouping is dynamic: as recent data ages, the Tuple Mover merges their partitions into month groups, and eventually into year groups.

For details, see the Administrator's Guide:
Partition Grouping

Hierarchical Partitioning

Data Analysis

This section contains information on updates to data analysis for Vertica Analytic Database 9.0.

Machine Learning for Predictive Analytics

Machine Learning for Predictive Analytics new features include:

- A new summary function, named \texttt{GET\_MODEL\_SUMMARY} that extends support from the existing \texttt{SUMMARIZE\_MODEL} function to read information from a JSON file.

- A new statistical summary function, named \texttt{SUMMARIZE\_NUMCOL}. This function provides a statistical summary of each feature in an input data set.

- Support for cross validation using the new \texttt{CROSS\_VALIDATE} function. Cross validation is useful for obtaining more accurate measurements, to verify that every sample from the original data set has the same chance of appearing in the training set and testing set. You can also use this function in hyper parameter selection to vary a particular parameter over a set of potential values, and then apply cross validation to each value of that parameter. Then, you can choose the value with the best performance as reported by cross validation.

- Support for one hot encoding, using the new \texttt{ONE\_HOT\_ENCODER\_FIT} and \texttt{APPLY\_ONE\_HOT\_ENCODER} functions. One hot encoding converts a categorical data set into a binary one, to indicate the presence or absence of that category. This is useful for algorithms that only accept numeric inputs, such as logistic regression.

- The ability to import and export your models to other Vertica clusters using the \texttt{IMPORT\_MODELS} and \texttt{EXPORT\_MODELS} functions.

For More Information

See Machine Learning for Predictive Analytics.
Apache Hadoop Integration

This section contains information on updates to Hadoop-integration information for Vertica Analytic Database 9.0.

Storage Locations Support hdfs URLs

You can now specify HDFS storage locations using URLs in the hdfs scheme, as you can for other Vertica Hadoop interfaces. Vertica falls back to using WebHDFS when necessary, so there is no disadvantage to using hdfs URLs.

You can still use webhdfs URLs for storage locations, but the documentation no longer shows them.

HCatalog Connector Supports Sentry for Hive Authentication

The HCatalog Connector now integrates with Sentry to manage authentication for Hive data. You must connect to Hive using HiveServer2 (the default), not WebHCat, to use this feature.

If Sentry uses ACL synchronization with HDFS, then no changes are required in Vertica. Of course, affected Vertica users must be granted access to the data in Hadoop. If Sentry does not use ACL synchronization, then you must allow the HCatalog Connector to access Hive data using the Vertica principal, and that principal must have access in HDFS. A new configuration parameter, EnableHCatImpersonation, controls how the HCatalog Connector accesses Hive.

For more information, see the HCatalog Connector section of Configuring Kerberos in Integrating with Apache Hadoop.

Alter Existing HCatalog Schema

The ALTER HCATALOG SCHEMA statement allows you to modify many of the parameters of an HCatalog schema. Previously, you had to recreate the schema to change parameters. For more information about HCatalog schemas, see Defining a Schema Using the HCatalog Connector.
COPY Supports Reading from S3 Buckets

You can now load or create external tables from Parquet and ORC data stored in S3 buckets. For more information, see Specifying COPY FROM Options in the Administrator's Guide.

Loading Data

This section contains information about loading data Vertica Analytic Database 9.0.

COPY Directly Supports S3

In previous versions, you loaded data from S3 buckets with syntax like the following:

```sql
=> COPY t WITH SOURCE S3(bucket='url');
```

COPY now directly supports S3 URLs for Parque, ORC, text, and delimited files, as in the following example:

```sql
=> COPY t FROM 's3://AWS_DataLake/sales.parquet' PARQUET;
```

If your data is in other formats, continue to use the older syntax.

For more information, see Specifying COPY FROM Options in the Administrator's Guide.

Eon Mode Beta

Vertica 9.0.0 allows you to operate your database in Eon Mode Beta. Doing so separates the computational processes from the storage layer of your database, thereby enabling rapid scaling of computational resources to accommodate variable demand workloads. Initial deployment of Eon Mode Beta is limited to Amazon Web Services.

Important: This functionality is currently in Beta. Vertica does not support running Eon Mode Beta in a production environment.

Vertica will not provide technical support for Eon Mode Beta users, but you can use the VerticaBeta Forum to ask and answer questions about operating your Vertica database in Eon Mode Beta.
To read the Eon Mode Beta documentation, go to Using Eon Mode Beta.

Management Console

This section contains information on updates to Management Console for Vertica Analytic Database 9.0.

Provision a Cluster and Database Using MC

Management Console with Provisioning is a new deployment of Management Console (MC) that allows you to provision, manage, and monitor Vertica on AWS resources. Launch Management Console with Provisioning and its associated AWS resources through the AWS Marketplace. After launching, use the new Cluster Creation wizard in MC to deploy Vertica cluster instances and create an Enterprise Mode or Eon Mode Beta database.

See Installing Using Management Console with Provisioning.
Manage Clusters in an AWS Environment

If you create a cluster on AWS resources using the new Cluster Creation wizard, the Cluster page in MC now shows additional AWS cluster and instance management options. You can perform AWS instance operations like stop, start, reboot and terminate on the whole cluster or individual instances, without switching to the AWS console.

See Viewing and Managing Your Cluster.

Fast Tasks

The Fast Tasks page offers a few important tasks to get you started managing your database through MC. This is the first page you see when you successfully create a new database using the new Cluster Creation wizard. To view the Fast Tasks page after cluster creation, click Fast Tasks in the Actions column of any available database on the MC home page.

See Fast Tasks.

Monitor Database Storage

The Infrastructure page in MC now provides a Storage View tab with a summary of the location and amount of data stored across your database. Use this view to monitor how much of your
storage capacity your databases are using.
See Monitoring Database Storage.

Monitor Shard Subscription Status

For databases running in Eon Mode Beta, the new Storage View tab also displays bar charts illustrating the subscription status of shards and nodes. Use the new Sharding Subscription and Node Subscription charts in MC to monitor the status of all shard subscriptions and how many nodes in your cluster are successfully subscribed to a shard.

See Monitoring Subscription Status in Eon Mode Beta.

For More Information

Using Management Console
Security and Authentication

Multi-Realm Support

Sophisticated customers typically have their users in a protected directory server (AD or Linux KDC) for their trusted realm (eg. EXAMPLE.COM) and they are reluctant to add service principals to it or export keytabs from there. Both of these happen to be steps required for configuring Vertica to use Kerberos.

So they resort to standing up another KDC with a different realm (eg. HADOOP.COM) to use with their Vertica and HDFS cluster. Later, a one-way trust is set up from EXAMPLE.COM to HADOOP.COM so that tickets obtained by employees in the realm EXAMPLE.COM can be used to request service tickets from the KDC in HADOOP.COM.

Multi-realm Support in Kerberos allows you to assign a different realm so that users from another realm can authenticate to Vertica.

SQL Functions and Statements

This section contains information on updates to SQL functions and statements for Vertica Analytic Database 9.0.

ALTER HCATALOG SCHEMA

When using the HCatalog Connector to access data in Hive, you can create a schema locally in Vertica. You can now use ALTER HCATALOG SCHEMA to modify that schema.

Partition Clause GROUP BY Option

Table partition clauses (specified in CREATE TABLE and ALTER TABLE) now support a GROUP BY clause that specifies how to consolidate partition keys into groups, where each group is identified by a unique partition group key. For details on usage, see Partition Grouping in the Administrator's Guide.
CALENDAR_HIERARCHY_DAY

To facilitate partition grouping, Vertica recommends that you use the new meta-function `CALENDAR_HIERARCHY_DAY`. This function can be specified in a table's partition clause to automatically group DATE partition keys into a hierarchy of years, months, and days. The Vertica Tuple Mover regularly evaluates partition keys against the current date, and merges partitions as needed into the appropriate year and month partition groups. For details on usage, see Hierarchical Partitioning in the Administrator's Guide.

Supported Data Types

UUID Data Type

Vertica now supports universally unique identifiers (UUIDs). It also provides the support function `UUID_GENERATE` for generating UUIDs that are based on high-quality randomness from /dev/urandom.

Performance Improvements

Identical Storage for Buddy Projections

INSERT and COPY operations that load directly into ROS now generate identical storage containers for buddy projections. Vertica creates storage containers and their metadata for one projection, and pipelines them to all buddy projections. This approach significantly reduces memory consumption and CPU workload, by eliminating redundant compression and sorting. It also minimizes network I/O.
Deprecated and Retired Functionality

This section describes the two phases Vertica follows to retire Vertica functionality:

- **Deprecated.** Vertica announces deprecated features and functionality in a major or minor release. Deprecated features remain in the product and are functional. Published release documentation announces deprecation on this page (see *Deprecated in This Release*, below). When users access this functionality, it can return informational messages about its pending removal.

- **Retired.** Vertica removes a feature in a major or minor release that follows the deprecation announcement. Users can no longer access the functionality, and this page is updated to verify removal (see *History*, below). Documentation that describes this functionality is removed, but remains in previous documentation versions.

### Deprecated in This Release

The following Vertica functionality was deprecated:

- `SUMMARIZE_MODEL` function is deprecated in favor of the new `GET_MODEL_SUMMARY` function in the Machine Learning for Predictive Analytics package.

- `RestrictSystemTable` parameter is deprecated.

- Vertica no longer supports SQL Server 2008.

- Vertica Pulse is deprecated.

### History

The following functionality has been deprecated or retired as indicated:

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Component</th>
<th>Deprecated Version</th>
<th>Removed Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertica Pulse</td>
<td>Server</td>
<td>9.0.1</td>
<td></td>
</tr>
<tr>
<td>Support for SQL Server 2008</td>
<td>Server</td>
<td>9.0.1</td>
<td>9.0.1</td>
</tr>
<tr>
<td>Functionality</td>
<td>Component</td>
<td>Deprecated Version</td>
<td>Removed Version</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>SUMMARIZE_MODEL function</td>
<td>Server</td>
<td>9.0.1</td>
<td></td>
</tr>
<tr>
<td>RestrictSystemTable parameter</td>
<td>Server</td>
<td>9.0.1</td>
<td></td>
</tr>
<tr>
<td>S3EXPORT() multipart parameter</td>
<td>Server</td>
<td>9.0</td>
<td></td>
</tr>
<tr>
<td>EnableStorageBundling parameter</td>
<td>Server</td>
<td>9.0</td>
<td></td>
</tr>
<tr>
<td>Machine Learning for Predictive Analytics package parameter key_columns for data preparation functions.</td>
<td>Server</td>
<td>9.0</td>
<td>9.0.1</td>
</tr>
<tr>
<td>Vertica meta-function DROP_PARTITION (superseded by DROP_PARTITIONS)</td>
<td>Server</td>
<td>9.0</td>
<td></td>
</tr>
<tr>
<td>Machine Learning for Predictive Analytics package parameter owner.</td>
<td>Server</td>
<td>8.1.1</td>
<td>9.0</td>
</tr>
<tr>
<td>RENAME_MODEL()</td>
<td>Server</td>
<td>8.1</td>
<td>8.1.1</td>
</tr>
<tr>
<td>DELETE_MODEL()</td>
<td>Server</td>
<td>8.1</td>
<td>8.1.1</td>
</tr>
<tr>
<td>Backup and restore --setupconfig command</td>
<td>Server</td>
<td>8.1</td>
<td></td>
</tr>
<tr>
<td>vbr --setupconfig command</td>
<td>Server</td>
<td>8.1</td>
<td></td>
</tr>
<tr>
<td>Ability to create a projection in a schema different from its anchor table</td>
<td>Server</td>
<td>8.0.1</td>
<td>8.1.1</td>
</tr>
<tr>
<td>SET_RECOVER_BY_TABLE(). Do not disable recovery by table.</td>
<td>Server</td>
<td>8.0.1</td>
<td></td>
</tr>
<tr>
<td>PreExcavatorReplicatedProjection()</td>
<td>Server</td>
<td>8.0.1</td>
<td></td>
</tr>
<tr>
<td>HDFS Connector</td>
<td>Server</td>
<td>8.0</td>
<td>9.0</td>
</tr>
<tr>
<td>Prejoin projections</td>
<td>Server</td>
<td>8.0</td>
<td></td>
</tr>
<tr>
<td>PROJECTIONS_USED column in system table V_MONITOR.QUERY_PROFILES</td>
<td>Optimizer</td>
<td>7.2.2</td>
<td>7.2.3</td>
</tr>
<tr>
<td>Administration Tools option --compat21</td>
<td>Server</td>
<td>7.2.1</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>Component</td>
<td>Deprecated Version</td>
<td>Removed Version</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>ext3 file system</strong></td>
<td>Server</td>
<td>7.2</td>
<td>8.0</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server (SLES) 11.0 Service Pack 3 (SP3) continues to support the ext3 file system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backup and restore overwrite configuration parameter</td>
<td>Server</td>
<td>7.2</td>
<td>7.2</td>
</tr>
<tr>
<td>Projection buddies with inconsistent sort order</td>
<td>Server</td>
<td>7.2</td>
<td></td>
</tr>
<tr>
<td>Backup.sh</td>
<td>Server</td>
<td>7.2</td>
<td>9.0</td>
</tr>
<tr>
<td>Restore.sh</td>
<td>Server</td>
<td>7.2</td>
<td>9.0</td>
</tr>
<tr>
<td>verticaConfig vbr configuration option</td>
<td>Server</td>
<td>7.1</td>
<td>7.2</td>
</tr>
<tr>
<td>JavaClassPathForUDx configuration parameter</td>
<td>Server</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td>ADD_LOCATION()</td>
<td>Server</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td>bwlimit</td>
<td>Server</td>
<td>7.1</td>
<td>9.0</td>
</tr>
<tr>
<td>EXECUTIONENGINE_PROFILES counters: file handles, memory allocated, and memory reserved</td>
<td>Server</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>MERGE_PARTITIONS()</td>
<td>Server</td>
<td>7.0</td>
<td>9.0</td>
</tr>
<tr>
<td>Version 6.0 vbr configuration mapping</td>
<td>Server</td>
<td>7.0</td>
<td>7.2</td>
</tr>
<tr>
<td>Administration Tools option check_spread</td>
<td>Server, clients</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>krb5 client authentication method</td>
<td>All clients</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>Use the Kerberos gss method for client authentication, instead of krb5. See Configuring Kerberos Authentication.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1oad library</td>
<td>Server</td>
<td>7.0</td>
<td>8.1.1</td>
</tr>
<tr>
<td>Functionality</td>
<td>Component</td>
<td>Deprecated Version</td>
<td>Removed Version</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>range-segmentation-clause</td>
<td>Server</td>
<td>6.1.1</td>
<td></td>
</tr>
<tr>
<td>scope parameter of CLEAR_PROFILING()</td>
<td>Server</td>
<td>6.1</td>
<td></td>
</tr>
<tr>
<td>IMPLEMENT_TEMP_DESIGN()</td>
<td>Server, clients</td>
<td>6.1</td>
<td></td>
</tr>
</tbody>
</table>
Vertica Concepts

This guide introduces the basic concepts to help you to effectively design, build, operate, and maintain a Vertica database. This document assumes that you are familiar with the basic concepts and terminology of relational database management systems and SQL.
The Vertica Approach

The Vertica Analytic Database consists of the following key features:

**Columnar Storage and Execution** - column stores offer significant gains in performance, I/O, storage footprint, and efficiency when it comes to analytic workloads. With columnar storage the query only reads the columns needed to answer the query.

**Real-time loading and querying** - with high query concurrency and the ability to simultaneously load new data into the system, Vertica can load data up to 10X faster than traditional row-store databases.

**Advanced Database Analytics** - a set of Advanced In-Database Analytics allows you to conduct the analytics computations closer to the data. This provides immediate results from a single place without having to extract data from a separate environment.

**Database Designer and Administration Tools** - these features allow you to tune and control Vertica with minimal administration effort. For more information see About Database Designer and Using the Administration Tools.

**Advanced Compression** - aggressive encoding and compression allows Vertica to dramatically improve analytic performance by reducing CPU, memory, and disk I/O at processing time. Vertica can reduce the original data size to up to 1/10th its original size.

**Massively Parallel Processing** - a robust and scalable parallel processing solution provides active redundancy, automatic replication, failover, and recovery.
Use Case

A mobile gaming company used to rely on a patchwork of technologies for data warehousing and business intelligence reporting. It took two to four hours to run a query on each game server. The search for a solution led the company to evaluate different companies for expanding its analytic capabilities. The Vertica implementation accomplished the following for the company:

- Queries were reduced from two to four hours to minutes or seconds
- Solution successfully met cloud deployment requirement
- Expanded data capacity from a few months to whole lifetime of data

This has led to better customer support by shortening response time to customer issues, as well as providing the ability to answer many more questions than before the Vertica implementation.
Getting Started

See Get Started With Vertica on our Community site for information on Vertica Analytic Database implementation.
Vertica Cluster Architecture

In Vertica, the physical architecture is designed to distribute physical storage and to allow parallel query execution over a potentially large collection of computing resources.

Hybrid Data Store

Vertica stores data on the database in two containers:

- Write Optimized Store (WOS) - stores data in memory without compression or indexing. You can use INSERT, UPDATE, and COPY to load data into WOS.

- Read Optimized Store (ROS) - stores data on disk. The data is segmented, sorted, and compressed for high optimization. You can load data directly into the ROS using the COPY statement.

The Tuple Mover moves data from the WOS (memory) to the ROS (disk) using the following processes:

- **Moveout** copies data from the WOS to the Tuple Mover and then to the ROS; data is sorted, encoded, and compressed into column files.

- **Mergeout** combines smaller ROS containers into larger ones to reduce fragmentation.

COPY can load data into WOS, where the Tuple Mover moves it to ROS. COPY can also specify loading data directly into the ROS. For more on load options, see Choosing a Load Method.

Column Storage

Vertica stores data in a column format so it can be queried for best performance. Compared to row-based storage, column storage reduces disk I/O making it ideal for read-intensive workloads. Vertica reads only the columns needed to answer the query. For example:

```
=> SELECT avg(price) FROM tickstore WHERE symbol = 'AAPL' and date = '5/31/13';
```

For this example query, a column store reads only three columns while a row store reads all columns:
Vertica uses encoding and compression to optimize query performance and save storage space.

Encoding converts data into a standard format. Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order. Encoding increases performance because there is less disk I/O during query execution. In addition, you can store more data in less space.

Compression transforms data into a compact format. Vertica uses several different compression methods and automatically chooses the best one for the data being compressed. Using compression, Vertica stores more data, provides more views, and uses less hardware than other databases. Using compression lets you keep much more historical data in physical storage.

The following shows compression using sorting and cardinality:
For more information, see Data Encoding and Compression.

## Clustering

Clustering supports scaling and redundancy. You can scale your database cluster by adding more hardware, and you can improve reliability by distributing and replicating data across your cluster.

Column data gets distributed across nodes in a cluster, so if one node becomes unavailable the database continues to operate. When a node is added to the cluster, or comes back online after being unavailable, it automatically queries other nodes to update its local data.
Projections

A projection consists of a set of columns with the same sort order, defined by a column to sort by or a sequence of columns by which to sort. Like an index or materialized view in a traditional database, a projection accelerates query processing. When you write queries in terms of the original tables, the query uses the projections to return query results.

Projections are distributed and replicated across nodes in your cluster, ensuring that if one node becomes unavailable, another copy of the data remains available. For more information, see K-Safety.

Automatic data replication, failover, and recovery provide for active redundancy, which increases performance. Nodes recover automatically by querying the system.

Logical and Physical Schema

Vertica stores information about database objects in the logical schema and the physical schema. The difference between the two schemas and how they relate to data storage is an important and unique aspect of the Vertica architecture.

A logical schema consists of objects such as tables, constraints, and views. Supports any relational schema design that you choose. A physical schema consists of collections of table columns called projections. A projection can contain some or all of the columns of a table.

Continuous Performance

Vertica queries and loads data continuously 24x7.

Concurrent loading and querying provides real-time views and eliminates the need for nightly load windows. On-the-fly schema changes allow you to add columns and projections without shutting down your database; Vertica manages updates while keeping the database available.
Terminology

It is helpful to understand the following terms when using Vertica:

Host

A computer system with a 32-bit (non-production use only) or 64-bit Intel or AMD processor, RAM, hard disk, and TCP/IP network interface (IP address and hostname). Hosts share neither disk space nor main memory with each other.

Instance

An instance of Vertica consists of the running Vertica process and disk storage (catalog and data) on a host. Only one instance of Vertica can be running on a host at any time.

Node

A host configured to run an instance of Vertica. It is a member of the database cluster. For a database to have the ability to recover from the failure of a node requires a database K-safety value of at least 1 (3+ nodes).

Cluster

The concept of Cluster in the Vertica Analytics Platform is a collection of hosts with the Vertica software packages (RPM or DEB) that are in one admin tools domain. You can access and manage a cluster from one admintools initiator host.

Database

A cluster of nodes that, when active, can perform distributed data storage and SQL statement execution through administrative, interactive, and programmatic user interfaces.

Note: Although you can define more than one database on a cluster, Vertica supports running only one database per cluster at a time.
Vertica in Eon Mode Beta Architecture

The architecture for Vertica in Eon Mode Beta separates the storage layer from the computation layer. This is different from Enterprise mode, which couples the storage and compute layers.

Depot

The depot creates a local copy of data associated with a shard to which a node subscribes. Each node in the cluster subscribes to a different shard or shards of the communal storage. The depot improves query performance by preventing some queries from making the long round trip to communal storage.

Data Storage

In Eon Mode Beta, COPY statements write to Read Optimized Store (ROS) files in a node's depot. The COPY statement segments, sorts, and compresses for high optimization. Before the statement commits, Vertica ships the ROS files to communal storage.

Note: Eon Mode Beta does not use the Write Optimized Storage (WOS).

The following diagram illustrates how data flows during a COPY statement.
Communal Storage

Instead of storing data locally, Eon Mode Beta uses a singular communal storage location for all data.
High Availability

Vertica provides high availability of the database using a RAID-like functionality. This provides the following mechanisms to ensure little to no downtime:

- Multiple copies of the same data on different nodes.
- Vertica continues to load and query data if a node is down.
- Vertica automatically recovers missing data by querying other nodes.

K-Safety

K-safety sets the fault tolerance in your Vertica database cluster. The value K represents the number of replicated data in the database cluster. These replicas allow other nodes to take over query processing for any failed nodes.

In Vertica, the value of K can be zero (0), one (1), or two (2). If a database with a K-safety of one (K=1) loses a node, the database continues to run normally. Potentially, the database could continue running if additional nodes fail, as long as at least one other node in the cluster has a copy of the failed node's data. Increasing K-safety to 2 ensures that Vertica can run normally if any two nodes fail. When the failed node or nodes return and successfully recover, they can participate in database operations again.

Note: If the number of failed nodes exceeds the K value, some the data may become unavailable. In this case, the database is considered unsafe and automatically shuts down. However, if every data segment is available on at least one functioning cluster node Vertica continues to run safely.

Potentially, up to half the nodes in a database with a K-safety of 1 could fail without causing the database to shut down. As long as the data on each failed node is available from another active node, the database continues to run.

Note: If half or more of the nodes in the database cluster fail, the database automatically shuts down even if all of the data in the database is available from replicas. This behavior prevents issues due to network partitioning.

Note: The physical schema design must meet certain requirements. To create designs that are K-safe, Vertica recommends using the Database Designer.
Buddy Projections

In order to determine the value of k-safety, Vertica creates buddy projections, which are copies of segmented projections distributed across database nodes. (See Projection Segmentation.) Vertica distributes segments that contain the same data to different nodes. This ensures that if a node goes down, all the data is available on the remaining nodes.

K-Safety Example

This diagram above shows a 5-node cluster with a K-safety level of 1. Each node contains buddy projections for the data stored in the next higher node (node 1 has buddy projections for node 2, node 2 has buddy projections for node 3, and so on). If any of the nodes fails the database continues to run, though with lower performance, since one of the nodes must handle its own workload and the workload of the failed node.

The diagram below shows a failure of Node 2. In this case, Node 1 handles processing for Node 2 since it contains a replica of node 2's data. Node 1 also continues to perform its own processing. The fault tolerance of the database falls from 1 to 0, since a single node failure could cause the database to become unsafe. In this example, if either Node 1 or Node 3 fails, the database becomes unsafe because not all of its data is available. If Node 1 fails, Node 2's data is no longer be available. If Node 3 fails, its data is no longer available, because node 2 is down and could not use the buddy projection. In this case, nodes 1 and 3 are considered critical nodes. In a database with a K-safety level of 1, the node that contains the buddy projection of a failed node, and the node whose buddy projections are on the failed node, always become critical nodes.
With Node 2 down, either node 4 or 5 could fail and the database still has all of its data available. The diagram below shows that if node 4 fails, node 3 can use its buddy projections to fill in for it. In this case, any further loss of nodes results in a database shutdown, since all the nodes in the cluster are now critical nodes. In addition, if one more node were to fail, half or more of the nodes would be down, requiring Vertica to automatically shut down, no matter if all of the data were available or not.

In a database with a K-safety level of 2, Node 2 and any other node in the cluster could fail and the database continues running. The diagram below shows that each node in the cluster contains buddy projections for both of its neighbors (for example, Node 1 contains buddy projections for Node 5 and Node 2). In this case, nodes 2 and 3 could fail and the database continues running. Node 1 could fill in for Node 2 and Node 4 could fill in for Node 3. Due to the requirement that half or more nodes in the cluster be available in order for the database to
continue running, the cluster could not continue running if node 5 failed, even though nodes 1 and 4 both have buddy projections for its data.

Note: Vertica requires that more than half of all nodes in a cluster must always be available; otherwise, it views the database as being in an unsafe state and shuts it down. Thus, in the previous example, the cluster cannot continue running if Node 5 fails, even though nodes 1 and 4 have buddy projections for its data.

Monitoring K-safety

You can access System Tables to monitor and log various aspects of Vertica operation. Use the SYSTEM table to monitor information related to K-safety, such as:

- **NODE_COUNT**: Number of nodes in the cluster
- **NODE_DOWN_COUNT**: Number of nodes in the cluster that are currently down
- **CURRENT_FAULT_TOLERANCE**: The K-safety level
High Availability With Projections

To ensure high availability and recovery for database clusters of three or more nodes, Vertica:

- Replicates small, unsegmented projections
- Creates buddy projections for large, segmented projections.

Replication (Unsegmented Projections)

When it creates projections, Database Designer replicates them, creating and storing duplicates of these projections on all nodes in the database.

Replication ensures:

- Distributed query execution across multiple nodes.
- High availability and recovery. In a K-safe database, replicated projections serve as buddy projections. This means that you can use a replicated projection on any node for recovery.

**Note:** We recommend you use Database Designer to create your physical schema. If you choose not to, be sure to segment all large tables across all database nodes, and replicate small, unsegmented table projections on all database nodes.

The following illustration shows two projections, B and C, replicated across a three node cluster.
Buddy Projections (Segmented Projections)

Vertica creates **buddy projections** which are copies of segmented projections that are distributed across database nodes (see Projection Segmentation.) Vertica distributes segments that contain the same data to different nodes. This ensures that if a node goes down, all the data is available on the remaining nodes. Vertica distributes segments to different nodes by using offsets. For example, segments that comprise the first buddy projection (A_BP1) are offset from projection A by one node, and segments from the second buddy projection (A_BP2) are offset from projection A by two nodes.

The following diagram shows the segmentation for a projection called A and its buddy projections, A_BP1 and A_BP2, for a three node cluster.

The following diagram shows how Vertica uses offsets to ensure that every node has a full set of data for the projection.
How Result Sets Are Stored

Vertica duplicates table columns on all nodes in the cluster to ensure high availability and recovery. Thus, if one node goes down in a K-Safe environment, the database continues to operate using duplicate data on the remaining nodes. Once the failed node resumes its normal operation, it automatically recovers its lost objects and data by querying other nodes.

Vertica compresses and encodes data to greatly reduce the storage space. It also operates on the encoded data whenever possible to avoid the cost of decoding. This combination of compression and encoding optimizes disk space while maximizing query performance.

Vertica stores table columns as projections. This enables you to optimize the stored data for specific queries and query sets. Vertica provides two methods for storing data:

- Projection segmentation is recommended for large tables (fact and large dimension tables)
- Replication is recommended for the rest of the tables.

High Availability with Fault Groups

Use fault groups to reduce the risk of correlated failures inherent in your physical environment. Correlated failures occur when two or more nodes fail as a result of a single failure. For example, such failures can occur due to problems with shared resources such as power loss, networking issues, or storage.

Vertica minimizes the risk of correlated failures by letting you define fault groups on your cluster. Vertica then uses the fault groups to distribute data segments across the cluster, so the database continues running if a single failure event occurs.

Note: If your cluster layout is managed by a single network switch, a switch failure would cause a single point of failure. Fault groups cannot help with single-point failures.

Vertica supports complex, hierarchical fault groups of different shapes and sizes. You can integrate fault groups with elastic cluster and large cluster arrangements to add cluster flexibility and reliability.
Making Vertica Aware of Cluster Topology with Fault Groups

You can also use fault groups to make Vertica aware of the topology of the cluster on which your Vertica database is running. Making Vertica aware of your cluster's topology is required when using Terrace Routing.

*Terrace routing* is a feature that can reduce the buffer requirements of large queries. Use terrace routing in situations where you have large queries and clusters with a large number of nodes. Without terrace routing, these situations would otherwise require excessive buffer space.

For more information about Terrace Routing, see Terraces Routing.

Automatic Fault Groups

When you configure a cluster of 120 nodes or more, Vertica automatically creates fault groups around control nodes. *Control nodes* are a subset of cluster nodes that manage spread (control messaging). Vertica places nodes that share a control node in the same fault group. See Large Cluster in the Administrator's Guide for details.

User-Defined Fault Groups

Define your own default groups if:

- Your cluster layout has the potential for correlated failures.
- You want to influence which cluster hosts manage control messaging.

Example Cluster Topology

The following diagram provides an example of hierarchical fault groups configured on a single cluster:
• Fault group FG–A contains nodes only.

• Fault group FG–B (parent) contains child fault groups FG–C and FG–D. Each child fault group also contain nodes.


How to Create Fault Groups

Before you define fault groups, you must have a thorough knowledge of your physical cluster layout. Fault groups require careful planning.

To define fault groups, create an input file of your cluster arrangement. Then, pass the file to a script supplied by Vertica, and the script returns the SQL statements you need to run. See Fault Groups in the Administrator's Guide for details.
Vertica Components

This section provides an overview of the components that make up Vertica. These components allow you to tune and control your Vertica Analytic Database with minimal effort. This eliminates the time and effort a database administrator of a typical database spends identifying issues.

Logical Schema

Design a logical schema for a Vertica database as you would for any SQL database. A logical schema consist of objects such as:

- schema
- table
- view
- Referential Integrity

Vertica supports any relational schema design that you choose. For more information, see Designing a Logical Schema in the Administrator's Guide.

Physical Schema

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns.

Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query. Vertica automatically refreshes these result sets with updated or new data.

Projections provide the following benefits:

- Compress and encode data to reduce storage space. Additionally, Vertica operates on the encoded data representation whenever possible to avoid the cost of decoding. This
combination of compression and encoding optimizes disk space while maximizing query performance.

- Facilitate distribution across the database cluster. Depending on their size, projections can be segmented or replicated across cluster nodes. For instance, projections for large tables can be segmented and distributed across all nodes. Unsegmented projections for small tables can be replicated across all nodes.

- Transparent to end-users. The Vertica query optimizer automatically picks the best projection to execute a given query.

- Provide high availability and recovery. Vertica duplicates table columns on at least K+1 nodes in the cluster. If one machine fails in a K-Safe environment, the database continues to operate using replicated data on the remaining nodes. When the node resumes normal operation, it automatically queries other nodes to recovers data and lost objects. For more information, see High Availability with Fault Groups and High Availability With Projections.

### Projection Types

A Vertica table typically has multiple projections, each defined to contain different content. Content for the projections of a given table can differ in scope and how it is organized. These differences can generally be divided into the following projection types:

#### Superprojections

A superprojection contains all the columns of a table. For each table in the database, Vertica requires a minimum of one superprojection.

Under certain conditions, Vertica automatically creates a table's superprojection immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. CREATE Projection can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

#### Query-Specific Projections

A query-specific projection is a projection that contains only the subset of table columns to process a given query. Query-specific projections significantly improve the performance of those queries for which they are optimized.
Aggregate Projections

Queries that include expressions or aggregate functions such as SUM and COUNT can perform more efficiently when they use projections that already contain the aggregated data. This is especially true for queries on large quantities of data.

Vertica provides three types of projections for storing data that is returned from aggregate functions or expressions:

- **Projection that contains expressions**: Projection with columns whose values are calculated from anchor table columns.

- **Live aggregate projection**: Projection that contains columns with values that are aggregated from columns in its anchor table. You can also define live aggregate projections that include user-defined transform functions.

- **Top-K projection**: Type of live aggregate projection that returns the top $k$ rows from a partition of selected rows. Create a Top-K projection that satisfies the criteria for a Top-K query.

For more information, see [Pre-Aggregating Data in Projections](#).

Projection Segmentation

You can define a projection to maintain its data on the cluster in two ways:

- Divided into multiple segments, or *segmented projections*

- Undivided storage units, or *unsegmented projections*

Segmented Projections

You typically create segmented projections for large fact tables. Vertica splits segmented projections into chunks (segments) of similar size and distributes these segments evenly across the cluster. System K-safety determines how many duplicates (*buddies*) of each segment are created and maintained on different nodes.

You create segmented projections with a `CREATE PROJECTION` statement that includes a `SEGMENTED BY` clause.

Projection segmentation achieves the following goals:
- Ensures high availability and recovery.
- Spreads the query execution workload across multiple nodes.
- Allows each node to be optimized for different query workloads.

Hash Segmentation

Vertica uses hash segmentation to segment large projections. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns typically meet these criteria, so they are often used as hash function arguments.

Unsegmented Projections

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are replicated without segmentation on all cluster nodes. You create unsegmented projections with a `CREATE PROJECTION` statement that includes the clause `UNSEGMENTED ALL NODES`. This clause specifies to create identical instances of the projection on all cluster nodes.

How Projections are Created

For each table in the database, Vertica requires a minimum of one projection, called a superprojection. A superprojection contains all columns in a given table. Vertica uses superprojections to ensure support for all queries and other DML operations.

Under certain conditions, Vertica automatically creates a table's superprojection immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. `CREATE PROJECTION` can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

While superprojections can support all queries on any table, they do not facilitate optimal execution of specific queries. Therefore, Vertica recommends that you run Database Designer on a representative sample of your data. Database Designer creates projections that optimize your database based on its data statistics and the queries you use, as follows:
1. Analyzes your logical schema, sample data and sample queries (optional).

2. Creates a physical schema, design (projections) in the form of a SQL script that can be deployed automatically or manually.

Database Designer creates designs that provide excellent query performance within physical constraints. Database Designer uses sophisticated strategies to provide excellent ad-hoc query performance while using disk space efficiently. If desired, you can also design custom projections.

Projection Definition Components

`CREATE  PROJECTION` defines a projection, as in the following example:

```sql
=> CREATE PROJECTION retail_sales_fact_p (  
    store_key ENCODING RLE,  
    pos_transaction_number ENCODING RLE,  
    sales_dollar_amount,  
    cost_dollar_amount )  
AS SELECT  
    store_key,  
    pos_transaction_number,  
    sales_dollar_amount,  
    cost_dollar_amount  
FROM store.store_sales_fact  
ORDER BY store_key  
SEGMENTED BY HASH(pos_transaction_number) ALL NODES;
```

A projection definition includes the following components:

- **Column List and Encoding**
- **Base Query**
- **Sort Order**
- **Segmentation**

**Column List and Encoding**

This portion of the SQL statement lists every column in the projection and defines the encoding for each column. Vertica supports encoded data, which helps query execution to incur less disk I/O.
CREATE PROJECTION retail_sales_fact_P (  
    store_key ENCODING RLE,  
    pos_transaction_number ENCODING RLE,  
    sales_dollar_amount,  
    cost_dollar_amount )

Base Query

A projection's base query clause identifies which columns to include in the projection.

AS SELECT  
    store_key,  
    pos_transaction_number,  
    sales_dollar_amount,  
    cost_dollar_amount

Sort Order

A projection's ORDER BY clause determines how to sort projection data. The sort order localizes logically grouped values so a disk read can identify many results at once. For maximum performance, do not sort projections on LONG VARBINARY and LONG VARCHAR columns. For more information see ORDER BY Clause

ORDER BY store_key

Segmentation

A projection's segmentation clause specifies how to distribute projection data across all nodes in the database. Even load distribution helps maximize access to projection data. For large tables, distribute projection data in segments with SEGMENTED BY HASH. For example:

SEGMENTED BY HASH(pos_transaction_number) ALL NODES;

For small tables, use the UNSEGMENTED keyword to replicate table data. Vertica creates identical copies of an unsegmented projection on all cluster nodes. Replication ensures high availability and recovery.

For maximum performance, do not segment projections on LONG VARBINARY and LONG VARCHAR columns.

For more information see Projection Segmentation.
Data Types

Vertica supports the following data types.

Structured Data

Structured data consists of all data that can be stored in a relational database. Vertica stores structured data in rows and columns. Structured data has relational keys that can be easily mapped to pre-designed fields.

Semi-structured Data

Semi-structured data does not reside in a relational database, but contains properties that allow it to be analyzed. Examples of semi-structured data are XML and JSON.

XML Example

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>22 3rd Street</streetAddress>
    <City>New York</City>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>
      <type>home</type>
      <number>212 555 5478</number>
    </phoneNumber>
    <phoneNumber>
      <type>mobile</type>
      <number>212 555 7841</number>
    </phoneNumber>
  </phoneNumbers>
  <gender>
    <type>Male</type>
  </gender>
</person>
```
JSON Example

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "22 3rd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    }
}
```

Unstructured Data

The majority of existing data is unstructured. Unstructured data does not have a pre-defined structure. It typically includes text and multi-media content, for example, emails, video files, and audio files. See Using Flex Tables for information on using unstructured data in Vertica.

Database

This section describes the following database elements:

Database Setup

This page provides an overview on setting up a Vertica database. For complete details see Configuring the Database.

Prepare SQL Scripts and Data Files

Prepare the following files before installing Vertica:

- Logical schema script
- Loadable data files
- **Load scripts**
- **Sample query script** (training set)

**Create the Database**

Create the database after installing Vertica on at least one host:

- Use the Administration Tools to:
  - Create a database
  - Connect to the database
- Use the Database Designer to design the physical schema.
- Use the `vsql` interactive interface to run SQL scripts that:
  - Create tables and constraints
  - Create projections

**Test the Empty Database**

- Test for sufficient projections using the sample query script
- Test the projections for K-safety

**Test the Partially-Loaded Database**

- Load the dimension tables
- Partially load the fact table
- Check system resource usage
- Check query execution times
- Check projection usage
Complete the Fact Table Load

- Monitor system usage
- Complete the fact table load

Set up Security

For security-related tasks, see Security and Authentication.

- [Optional] Set up SSL
- [Optional] Set up client authentication
- Set up database users and privileges

Set up Incremental Loads

Set up periodic (trickle) loads, see Trickle Loading Data.

Database Connections

You can connect to a Vertica database in the following ways:

- Interactively using the vsq1 client, as described in Using vsq1 in the Administrator's Guide.

  vsq1 is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

  You can run vsq1 on any node within a database. To start vsq1, use the Administration Tools or the shell command described in Using vsq1.

- Programmatically using the JDBC driver provided by Vertica, as described in Programming JDBC Client Applications in Connecting to Vertica.

  An abbreviation for Java Database Connectivity, JDBC is a call-level application programming interface (API) that provides connectivity between Java programs and data sources (SQL
databases and other non-relational data sources, such as spreadsheets or flat files). JDBC is included in the Java 2 Standard and Enterprise editions.

- Programmmatically using the ODBC driver provided by Vertica, as described in Programming ODBC Client Applications in Connecting to Vertica.

An abbreviation for Open DataBase Connectivity, ODBC is a standard application programming interface (API) for access to database management systems.

- Programatically using the ADO.NET driver provided by Vertica, as described in Programming ADO.NET Applications in Connecting to Vertica.

The Vertica driver for ADO.NET allows applications written in C# and Visual Studio to read data from, update, and load data into Vertica databases. It provides a data adapter that facilitates reading data from a database into a data set, and then writing changed data from the data set back to the database. It also provides a data reader (VerticaDataReader) for reading data and autocommit functionality for committing transactions automatically.

- Programatically using Perl and the DBI driver, as described in Programming Perl Client Applications in Connecting to Vertica.

Perl is a free, stable, open source, cross-platform programming language licensed under its Artistic License, or the GNU General Public License (GPL).

- Programatically using Python and the Vertica Python Client or the pyodbc driver, as described in Programming Python Client Applications in Connecting to Vertica.

Python is a free, agile, object-oriented, cross-platform programming language designed to emphasize rapid development and code readability.

Vertica recommends that you deploy Vertica as the only active process on each machine in the cluster and connect to it from applications on different machines. Vertica expects to use all available resources on the machine, and to the extent that other applications are also using these resources, suboptimal performance could result.

Database Security

Vertica secures access to the database and its resources by enabling you to control user access to the database and which tasks users are authorized to perform. See Security and Authentication.
Database Designer

Vertica’s Database Designer is a tool that:

- Analyzes your logical schema, sample data, and, optionally, your sample queries.
- Creates a Physical Schema design that can be deployed automatically or manually.
- Can be used by anyone without specialized database knowledge. Even business users can run Database Designer.
- Can be run and re-run any time for additional optimization without stopping the database.

Run the DBD

Run the Database Designer in one of the following ways:

- With the Management Console, as described in Using Management Console to Create a Design
- Programmatically, using the steps described in About Running Vertica Programmatically.
- With the Administration Tools by selecting Configuration Menu > Run Database Designer. For details, see Using the Administration Tools to Create a Design

Use the Database Designer to create one of the following types of designs:

- A comprehensive design that allows you to create new projections for all tables in your database.
- An incremental design that creates projections for all tables referenced in the queries you supply.

Database Designer benefits include:

- Accepting up to 100 queries in the query input file for an incremental design.
- Accepting unlimited queries for a comprehensive design.
- Producing higher quality designs by considering UPDATE and DELETE statements.
In most cases, the designs created by Database Designer provide optimal query performance within physical constraints. Database Designer uses sophisticated strategies to provide optimal query performance and data compression.

**See Also**

- Physical Schema
- Creating a Database Design

**Data Loading**

The SQL Data Manipulation Language (DML) commands INSERT, UPDATE, and DELETE perform the same functions in Vertica as they do in row-oriented databases. These commands follow the SQL-92 transaction model and can be intermixed.

Use the **COPY** statement for bulk loading data. COPY reads data from text files or data pipes and inserts it into WOS (memory) or directly into the ROS (disk). COPY can load compressed formats such as GZIP and LZO. COPY automatically commits itself and any current transaction but is not atomic; some rows could be rejected. Note that COPY does not automatically commit when copying data into temporary tables.

You can use the COPY statement's NO COMMIT option to prevent COPY from committing a transaction when it finishes copying data. This allows you to ensure the data in the bulk load is either committed or rolled back at the same time. Also, combining multiple smaller data loads into a single transaction allows Vertica to load the data more efficiently. See the **COPY statement** in the SQL Reference Manual for more information.

You can use multiple, simultaneous database connections to load and/or modify data.

For more information about bulk loading, see [Bulk Loading Data](#).

**Workload Management**

Vertica's resource management scheme allows diverse, concurrent workloads to run efficiently on the database. For basic operations, Vertica pre-configures the built-in **GENERAL pool** based on RAM and machine cores. You can customize the General pool to handle specific concurrency requirements.

You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally assign each database user to use a specific resource pool, which controls memory resources used by their requests.
User-defined pools are useful if you have competing resource requirements across different classes of workloads. Example scenarios include:

- A large batch job takes up all server resources, leaving small jobs that update a web page without enough resources. This can degrade user experience.

  In this scenario, create a resource pool to handle web page requests and ensure users get resources they need. Another option is to create a limited resource pool for the batch job, so the job cannot use up all system resources.

- An application has lower priority than other applications and you want to limit the amount of memory and number of concurrent users for the low-priority application.

  In this scenario, create a resource pool with an upper limit on the query's memory and associate the pool with users of the low-priority application.

For more information, see Managing Workload Resources in the Administrator's Guide.
Management Console

Management Console (MC) is a user-friendly performance monitoring and management tool that provides a unified view of your Vertica database operations. Using a browser, you can create, import, manage, and monitor one or more databases and their associated clusters. You can also create and manage MC users. You can then map the MC users to a Vertica database and manage them through the MC interface.

What You Can Do with Management Console

<table>
<thead>
<tr>
<th>Create...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A database cluster on hosts that do not have Vertica installed</td>
</tr>
<tr>
<td>Multiple Vertica databases on one or more clusters from a single point of control</td>
</tr>
<tr>
<td>MC users and grant them access to MC and databases managed by MC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configure...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database parameters and user settings dynamically</td>
</tr>
<tr>
<td>Resource pools</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monitor...</th>
</tr>
</thead>
<tbody>
<tr>
<td>License usage and conformance</td>
</tr>
<tr>
<td>Dynamic metrics about your database cluster</td>
</tr>
<tr>
<td>Resource pools</td>
</tr>
<tr>
<td>User information and activity on MC</td>
</tr>
<tr>
<td>Alerts by accessing a single message box of alerts for all managed databases</td>
</tr>
<tr>
<td>Recent databases and clusters through a quick link</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Import or Export...</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Troubleshoot...</th>
</tr>
</thead>
</table>
Management Console provides some, but not all, the functionality that Administration Tools provides. Management Console also includes extended functionality not available in admintools. This additional functionality includes a graphical view of your Vertica database and detailed monitoring charts and graphs. See Administration Tools and Management Console in the Administrator's Guide for more information.

Getting MC

Download the Vertica server RPM and the MC package from myVertica Portal. You then have two options:

- Install Vertica and MC at the command line and import one or more Vertica database clusters into the MC interface
- Install Vertica directly through MC

See the Installation Guide for details.

What You Need to Know

If you plan to use MC, review the following topics in the Administrator's Guide:
If you want to ... | See ...
--- | ---
Create a new, empty Vertica database | Create a Database on a Cluster
Import an existing Vertica database cluster into MC | Managing Database Clusters
Understand how MC users differ from database users | About MC Users
Read about the MC privilege model | About MC Privileges and Roles
Create new MC users | Creating an MC User
Grant MC users privileges on one or more Vertica databases managed by MC | Granting Database Access to MC Users
Use Vertica functionality through the MC interface | Using Management Console
Monitor MC and Vertica databases managed by MC | Monitoring Vertica Using Management Console
Monitor and configure Resource Pools | Monitoring Resource Pools

Management Console Architecture

MC accepts HTTP requests from a client web browser, gathers information from the Vertica database cluster, and returns that information to the browser for monitoring.

MC Components

The primary components that drive Management Console are an application/web server and agents that get installed on each node in the Vertica cluster.

The following diagram is a logical representation of MC, the MC user's interface, and the database cluster nodes.
Application/web Server

The application server hosts MC's web application and uses port 5450 for node-to-MC communication and to perform the following:

- Manage one or more Vertica database clusters
- Send rapid updates from MC to the web browser
- Store and report MC metadata, such as alerts and events, current node state, and MC users, on a lightweight, embedded (Derby) database
- Retain workload history

MC Agents

MC agents are internal daemon process that run on each Vertica cluster node. The default agent port, 5444, must be available for MC-to-node and node-to-node communications. Agents monitor MC-managed Vertica database clusters and communicate with MC to provide the following functionality:

- Provide local access, command, and control over database instances on a given node, using functionality similar to Administration Tools.
- Report log-level data from the Administration Tools and Vertica log files.
- Cache details from long-running jobs—such as create/start/stop database operations—that you can view through your browser.
- Track changes to data-collection and monitoring utilities and communicate updates to MC.
Communicate between all cluster nodes and MC through a webhook subscription, which automates information sharing and reports on cluster-specific issues like node state, alerts, and events.

See Also

- Monitoring Using MC

Management Console Security

The Management Console (MC) manages multiple Vertica clusters, all which might have different levels and types of security, such as user names and passwords and LDAP authentication. You can also manage MC users who have varying levels of access across these components.

Open Authorization and SSL

Management Console (MC) uses a combination of OAuth (Open Authorization), Secure Socket Layer (SSL), and locally-encrypted passwords to secure HTTPS requests between a user's browser and MC, and between MC and the agents. Authentication occurs through MC and between agents within the cluster. Agents also authenticate and authorize jobs.

The MC configuration process sets up SSL automatically, but you must have the openssl package installed on your Linux environment first.

See the following topics in the Administrator's Guide for more information:

- SSL Overview
- TLS/SSL Server Authentication
- Generating Certificates and Keys for MC
- Importing a New Certificate to MC
User Authentication and Access

MC provides two user authentication methods, LDAP or MC. You can use only one method at a time. For example, if you chose LDAP, all MC users will be authenticated against your organization's LDAP server.

You set LDAP authentication up through MC Settings > Authentication on the MC interface.

Note: MC uses LDAP data for authentication purposes only. It does not modify user information in the LDAP repository.

The MC authentication method stores MC user information internally and encrypts passwords. These MC users are not system (Linux) users. They are accounts that have access to MC and, optionally, to one or more MC-managed Vertica databases through the MC interface.

Management Console also has rules for what users can see when they sign in to MC from a client browser. These rules are governed by access levels, each of which is made up of a set of roles.

See Also

- About MC Users
- About MC Privileges and Roles
- Creating an MC User

Management Console Home Page

The MC Home page is the entry point to all MC-managed Vertica database clusters and MC users. User access levels determine what a user can see on the MC Home page. Layout and navigation are described in Using Management Console.

Administration Tools

The Vertica Administration tools allow you to easily perform administrative tasks. You can perform most Vertica database administration tasks with Administration Tools.
Run Administration Tools using the Database Administrator account on the Administration host, if possible. Make sure that no other Administration Tools processes are running.

If the Administration host is unresponsive, run Administration Tools on a different node in the cluster. That node permanently takes over the role of Administration host.

Any user can view the man page available for admintools. Enter the following:

```
man admintools
```

### Running Administration Tools

As dbadmin user, you can run administration tools. The syntax follows:

```
/opt/vertica/bin/admintools [--debug ][
   {-h | --help }
   | { -a | --help_all}
   | { -t | --tool } name_of_tool[ options]
]
```

#### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--debug</strong></td>
<td>If you include the debug option, Vertica logs debug information.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You can specify the debug option with or without naming a specific tool. If you specify debug with a specific tool, Vertica logs debug information during tool execution. If you do not specify a tool, Vertica logs debug information when you run tools through the admintools user interface.</td>
</tr>
<tr>
<td><strong>-h</strong></td>
<td>Outputs abbreviated help.</td>
</tr>
<tr>
<td><strong>--help</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-a</strong></td>
<td>Outputs verbose help, which lists all command-line sub-commands and options.</td>
</tr>
<tr>
<td><strong>--help_all</strong></td>
<td></td>
</tr>
<tr>
<td>**{ -t</td>
<td>--tool }** name_of_tool [options]</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Enter admintools -h to see the list of tools available. Enter admintools -t name_of_tool --help to review a specific tool's options.</td>
</tr>
</tbody>
</table>

An unqualified admintools command displays the Main Menu dialog box.
If you are unfamiliar with this type of interface, read Using the Administration Tools Interface

**First Login as Database Administrator**

The first time you log in as the Database Administrator and run the Administration Tools, the user interface displays.

1. In the end-user license agreement (EULA) window, type accept to proceed.

   A window displays, requesting the location of the license key file you downloaded from the Vertica Web site. The default path is /tmp/vlicense.dat.

2. Type the absolute path to your license key (for example, /tmp/vlicense.dat) and click OK.

**Between Dialogs**

While the Administration Tools are working, you see the command line processing in a window similar to the one shown below. Do not interrupt the processing.
SQL in Vertica

Vertica offers a robust set of SQL elements that allow you to manage and analyze massive volumes of data quickly and reliably. Vertica uses the following:

**SQL language elements**, including:

- Keywords and Reserved Words
- Identifiers
- Literals
- Operators
- Expressions
- Predicates
- Hints

**SQL data types**, including:

- Binary
- Boolean
- Character
- Date/Time
Long

Numeric

**SQL functions** including Vertica-specific functions that take advantage of Vertica's unique column-store architecture. For example, call `ANALYZE_STATISTICS` to collect and aggregate a variable amount of sample data for statistical analysis.

**SQL statements** that let you write robust queries to quickly return large volumes of data.

### About Query Execution

When you submit a query, the query optimizer quickly chooses the projections to use, optimizes and plans the query execution, and logs the SQL statement to its log. This planning results in an query plan, which maps out the steps the query performs. You can view a query plan in by embedding the query in an `EXPLAIN` statement; you can also view it in the Management Console.

The optimizer breaks down the query plan into smaller plans and distributes them to **Executor Node**

In the final stages of query plan execution, the initiator node performs the following tasks:

- Combines results in a grouping operation.
- Merges multiple sorted partial result sets from all the executors.
- Formats the results to return to the client.

For detailed information about writing and executing queries, see **Queries** in Analyzing Data.

### Historical Queries

Vertica can run a query from a snapshot of the database taken at a specific date and time or at a specific epoch. For details, see **Historical Queries** in Analyzing Data.

### Snapshot Isolation Mode

You can run any SQL query in snapshot isolation mode to obtain the fastest possible execution. Snapshot isolation mode is an **historical query** that gets data from the latest epoch:
AT EPOCH LATEST SELECT...

The query returns all data from the latest epoch, without holding a lock or blocking write operations. Thus, the query can access and return data that was loaded by other users up to (but no more than) a specific number of minutes before it executes.
International Languages and Character Sets

This section describes how Vertica handles internationalization and character sets.

Unicode Character Encoding

Vertica supports Unicode Transformation Format-8, or UTF8, where 8 equals 8-bit. UTF-8 is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard. Initial encoding of byte codes and character assignments for UTF-8 coincides with ASCII. Thus, UTF8 requires little or no change for software that handles ASCII but preserves other values.

Vertica database servers expect to receive all data in UTF-8, and Vertica outputs all data in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. JDBC and ADO.NET APIs operate on data in UTF-16. Client drivers automatically convert data to and from UTF-8 when sending to and receiving data from Vertica using API calls. The drivers do not transform data loaded by executing a COPY or COPY LOCAL statement.


Locales

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate GROUP BY and ORDER BY clauses, joins, and the analytic ORDER BY clause.

The default locale for a Vertica database is en_US@collation=binary (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default en_US@collation=binary collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to en_US_POSIX. You can set the locale back to the default locale and collation by issuing the vsqi meta-command `\locale`. For example:
You can set locale through ODBC, JDBC, and ADO.net.

See the following topics in the Administrator's Guide for details:

- Implement Locales for International Data Sets
- Supported Locales in the Appendix

### String Functions

Vertica provides string functions to support internationalization. Unless otherwise specified, these string functions can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of characters. This is accomplished by adding "USING OCTETS" and "USING CHARACTERS" (default) as a parameter to the function.

See String Functions for details.

### Character String Literals

By default, string literals ('...') treat back slashes literally, as specified in the SQL standard.

**Tip:** If you have used previous releases of Vertica and you do not want string literals to treat back slashes literally (for example, you are using a back slash as part of an escape sequence), you can turn off the StandardConformingStrings configuration parameter. See Internationalization Parameters in the Administrator's Guide. You can also use the EscapeStringWarning parameter to locate back slashes which have been incorporated into string literals, in order to remove them.

See Character String Literals for details.
Welcome to Installing Vertica. Read this guide to learn how to prepare for and install the Vertica server. This guide also provides instructions for installing the Vertica Management Console.

For information about installing client drivers, see Client Drivers.
Prerequisites

- This document assumes that you have become familiar with the concepts discussed in Vertica Concepts.

- To perform the procedures described in this document, you must have root password or sudo access (for all commands) for all nodes in your cluster.
Installation Overview and Checklist

This page provides an overview of installation tasks. Carefully review and follow the instructions in all sections in this topic.

Important Notes

- Vertica supports only one running database per cluster.

- Vertica supports installation on one, two, or multiple nodes. The steps for Installing Vertica are the same, no matter how many nodes are in the cluster.

- Prerequisites listed in Before You Install Vertica are required for all Vertica configurations.

- Only one instance of Vertica can be running on a host at any time.

- To run the install_vertica script, as well as adding, updating, or deleting nodes, you must be logged in as root, or sudo as a user with all privileges. You must run the script for all installations, including upgrades and single-node installations.

Installation Scenarios

The four main scenarios for installing Vertica on hosts are:

- A single node install, where Vertica is installed on a single host as a localhost process. This form of install cannot be expanded to more hosts later on and is typically used for development or evaluation purposes.

- Installing to a cluster of physical host hardware. This is the most common scenario when deploying Vertica in a testing or production environment.

- Installing on Amazon Web Services (AWS). When you choose the recommended Amazon Machine Image (AMI), Vertica is installed when you create your instances. For the AWS specific installation procedure, see Installing and Running Vertica on AWS: The Detailed Procedure rather than the using the steps for installation and upgrade that appear in this guide.
Installing to a local cluster of virtual host hardware. Also similar to installing on physical hosts, but with network configuration differences.

Before You Install

Before You Install Vertica describes how to construct a hardware platform and prepare Linux for Vertica installation.

These preliminary steps are broken into two categories:

- Configuring Hardware and Installing Linux
- Configuring the Network

Install or Upgrade Vertica

Once you have completed the steps in the Before You Install Vertica section, you are ready to run the install script.

Installing Vertica describes how to:

- Back up any existing databases.
- Download and install the Vertica RPM package.
- Install a cluster using the install vertica script.
- [Optional] Create a properties file that lets you install Vertica silently.

Note: This guide provides additional manual procedures in case you encounter installation problems.

- Upgrading Vertica describes how to upgrade to a more recent version of the software.

Note: If you are upgrading your Vertica license, refer to Managing Licenses in the Administrator's Guide.
Post-Installation Tasks

After You Install Vertica describes subsequent steps to take after you've run the installation script. Some of the steps can be skipped based on your needs:

- Install the license key.
- Verify that kernel and user parameters are correctly set.
- Install the vsql client application on non-cluster hosts.
- Resolve any SLES 11.3 issues during spread configuration.
- Use the Vertica documentation online, or download and install Vertica documentation. Find the online documentation and documentation packages to download at http://my.vertica.com/docs.
- Install client drivers.
- Extend your installation with Vertica packages.
- Install or upgrade the Management Console.

Get started!

- Read the Concepts Guide for a high-level overview of the Vertica Analytics Platform.
- Proceed to the Installing and Connecting to the VMart Example Database in Getting Started, where you will be guided through setting up a database, loading sample data, and running sample queries.
About Linux Users Created by Vertica and Their Privileges

This topic describes the Linux accounts that the installer creates and configures so Vertica can run. When you install Vertica, the installation script optionally creates the following Linux user and group:

- dbadmin—Administrative user
- verticadba—Group for DBA users

dbadmin and verticadba are the default names. If you want to change what these Linux accounts are called, you can do so using the installation script. See Installing Vertica with the Installation Script for details.

Before You Install Vertica

See the following topics for more information:

- Installation Overview and Checklist
- General Hardware and OS Requirements and Recommendations

When You Install Vertica

The Linux dbadmin user owns the database catalog and data storage on disk. When you run the install script, Vertica creates this user on each node in the database cluster. It also adds dbadmin to the Linux dbadmin and verticadba groups, and configures the account as follows:

- Configures and authorizes dbadmin for passwordless SSH between all cluster nodes. SSH must be installed and configured to allow passwordless logins. See Enable Secure Shell (SSH) Logins.

- Sets the dbadmin user's BASH shell to /bin/bash, required to run scripts, such as install_vertica and the Administration Tools.

- Provides read-write-execute permissions on the following directories:
- /opt/vertica/*

- /home/dbadmin—the default directory for database data and catalog files (configurable through the install script)

Note: The Vertica installation script also creates a Vertica database superuser named dbadmin. They share the same name, but they are not the same; one is a Linux user and the other is a Vertica user. See Database Administration User in the Administrator's Guide for information about the database superuser.

After You Install Vertica

Root or sudo privileges are not required to start or run Vertica after the installation process completes.

The dbadmin user can log in and perform Vertica tasks, such as creating a database, installing/changing the license key, or installing drivers. If dbadmin wants database directories in a location that differs from the default, the root user (or a user with sudo privileges) must create the requested directories and change ownership to the dbadmin user.

Vertica prevents administration from users other than the dbadmin user (or the user name you specified during the installation process if not dbadmin). Only this user can run Administration Tools.

See Also

- Installation Overview and Checklist
- Before You Install Vertica
- Platform Requirements and Recommendations
- Enable Secure Shell (SSH) Logins
Before You Install Vertica

Complete all of the tasks in this section before you install Vertica. When you have completed this section, proceed to Installing Vertica.
Platform Requirements and Recommendations

You must verify that your servers meet the platform requirements described in Supported Platforms. The Supported Platforms topics detail supported versions for the following:

- OS for Server and Management Console (MC)
- Supported Browsers for MC
- Vertica driver compatibility
- R
- Hadoop
- Various plug-ins

BASH Shell

All shell scripts included in Vertica must run under the BASH shell. If you are on a Debian system, then the default shell can be DASH. DASH is not supported. Change the shell for root and for the dbadmin user to BASH with the chsh command.

For example:

```bash
# getent passwd | grep root
root:x:0:0:root:/root:/bin/dash

# chsh
Changing shell for root.
New shell [/bin/bash]: /bin/bash
Shell changed.
```

Then, as root, change the symbolic link for `/bin/sh` from `/bin/dash` to `/bin/bash`:

```bash
# rm /bin/sh
# ln -s /bin/bash /bin/sh
```

Log out and back in for the change to take effect.
Install the Latest Vendor Specific System Software

Install the latest vendor drivers for your hardware. For Hewlett Packard Servers, update to the latest versions for:

- HP ProLiant Smart Array Controller Driver (cciss)
- Smart Array Controller Firmware
- HP Array Configuration Utility (HP ACU CLI)

Data Storage Recommendations

- All internal drives connect to a single RAID controller.
- The RAID array should form one hardware RAID device as a contiguous /data volume.

Validation Utilities

Vertica provides several validation utilities that validate the performance on prospective hosts. The utilities are installed when you install the Vertica RPM, but you can use them before you run the install_vertica script. See Validation Scripts for more details on running the utilities and verifying that your hosts meet the recommended requirements.

General Hardware and OS Requirements and Recommendations

Hardware Recommendations

The Vertica Analytics Platform is based on a massively parallel processing (MPP), shared-nothing architecture, in which the query processing workload is divided among all nodes of the Vertica database. OpenText highly recommends using a homogeneous hardware configuration for your Vertica cluster; that is, each node of the cluster should be similar in CPU, clock speed, number of cores, memory, and operating system version.
Note that OpenText has not tested Vertica on clusters made up of nodes with disparate hardware specifications. While it is expected that a Vertica database would functionally work in a mixed hardware configuration, performance will most certainly be limited to that of the slowest node in the cluster.

Detailed hardware recommendations are available in Recommendations for Sizing Vertica Nodes and Clusters (formerly the Vertica Hardware Planning Guide).

**Platform OS Requirements**

**Important:** Deploy Vertica as the only active process on each host—other than Linux processes or software explicitly approved by Vertica. Vertica cannot be collocated with other software. Remove or disable all non-essential applications from cluster hosts.

You must verify that your servers meet the platform requirements described in Vertica Server and Vertica Management Console.

**Verify Sudo**

Vertica uses the sudo command during installation and some administrative tasks. Ensure that sudo is available on all hosts with the following command:

```
# which sudo
/usr/bin/sudo
```

If sudo is not installed, browse to the Sudo Main Page and install sudo on all hosts.

When you use sudo to install Vertica, the user that performs the installation must have privileges on all nodes in the cluster.

Configuring sudo with privileges for the individual commands can be a tedious and error-prone process; thus, the Vertica documentation does not include every possible sudo command that you can include in the sudoers file. Instead, Vertica recommends that you temporarily elevate the sudo user to have all privileges for the duration of the install.

**Note:** See the sudoers and visudo man pages for the details on how to write/modify a sudoers file.

To allow root sudo access on all commands as any user on any machine, use visudo as root to edit the /etc/sudoers file and add this line:

```
## Allow root to run any commands anywhere
root    ALL=(ALL) ALL
```
After the installation completes, remove (or reset) sudo privileges to the pre-installation settings.
Prepare Disk Storage Locations

You must create and specify directories in which to store your catalog and data files (physical schema). You can specify these locations when you install or configure the database, or later during database operations. Both the catalog and data directories must be owned by the database administrator.

The directory you specify for database catalog files (the catalog path) is used across all nodes in the cluster. For example, if you specify /home/catalog as the catalog directory, Vertica uses that catalog path on all nodes. The catalog directory should always be separate from any data file directories.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

The data path you designate is also used across all nodes in the cluster. Specifying that data should be stored in /home/data, Vertica uses this path on all database nodes.

Do not use a single directory to contain both catalog and data files. You can store the catalog and data directories on different drives, which can be either on drives local to the host (recommended for the catalog directory) or on a shared storage location, such as an external disk enclosure or a SAN.

Before you specify a catalog or data path, be sure the parent directory exists on all nodes of your database. Creating a database in admintools also creates the catalog and data directories, but the parent directory must exist on each node.

You do not need to specify a disk storage location during installation. However, you can do so by using the --data-dir parameter to the install_vertica script. See Specifying Disk Storage Location During Installation

Disk Space Requirements for Vertica

In addition to actual data stored in the database, Vertica requires disk space for several data reorganization operations, such as mergeout and managing nodes in the cluster. For best results, Vertica recommends that disk utilization per node be no more than sixty percent (60%) for a K-Safe=1 database to allow such operations to proceed.
In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they cannot be completed in memory (RAM). Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed (known as temp space) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By default, any unused disk space on the data disk can be used as temp space. However, Vertica recommends provisioning temp space separate from data disk space.

See Also

Configuring Disk Usage to Optimize Performance.
Configuring the Network

This group of steps involve configuring the network. These steps differ depending on your installation scenario. A single node installation requires little network configuration, since the single instance of the Vertica server does not need to communicate with other nodes in a cluster. For cluster and cloud install scenarios, you must make several decisions regarding your configuration.

Vertica supports server configuration with multiple network interfaces. For example, you might want to use one as a private network interface for internal communication among cluster hosts (the ones supplied via the --hosts option to install_vertica) and a separate one for client connections.

Important: Vertica performs best when all nodes are on the same subnet and have the same broadcast address for one or more interfaces. A cluster that has nodes on more than one subnet can experience lower performance due to the network latency associated with a multi-subnet system at high network utilization levels.

Important Notes

- Network configuration is exactly the same for single nodes as for multi-node clusters, with one special exception. If you install Vertica on a single host machine that is to remain a permanent single-node configuration (such as for development or Proof of Concept), you can install Vertica using localhost or the loopback IP (typically 127.0.0.1) as the value for --hosts. Do not use the hostname localhost in a node definition if you are likely to add nodes to the configuration later.

- If you are using a host with multiple network interfaces, configure Vertica to use the address which is assigned to the NIC that is connected to the other cluster hosts.

- Use a dedicated gigabit switch. If you do not performance could be severely affected.

- Do not use DHCP dynamically-assigned IP addresses for the private network. Use only static addresses or permanently-leased DHCP addresses.
Optionally Run Spread on Separate Control Network

If your query workloads are network intensive, you can use the `--control-network` parameter with the `install_vertica` script (see Installing Vertica with the Installation Script) to allow spread communications to be configured on a subnet that is different from other Vertica data communications.

The `--control-network` parameter accepts either the default value or a broadcast network IP address (for example, `192.168.10.255`).

Configure SSH

- Verify that root can use Secure Shell (SSH) to log in (ssh) to all hosts that are included in the cluster. SSH (SSH client) is a program for logging into a remote machine and for running commands on a remote machine.

- If you do not already have SSH installed on all hosts, log in as root on each host and install it before installing Vertica. You can download a free version of the SSH connectivity tools from OpenSSH.

- Make sure that `/dev/pts` is mounted. Installing Vertica on a host that is missing the mount point `/dev/pts` could result in the following error when you create a database:

```
TIMEOUT ERROR: Could not login with SSH. Here is what SSH said:
Last login: Sat Dec 15 18:05:35 2007 from v_vmart_node0001
```

Allow Passwordless SSH Access for the Dbadmin User

The dbadmin user must be authorized for passwordless ssh. In typical installs, you won't need to change anything; however, if you set up your system to disallow passwordless login, you'll need to enable it for the dbadmin user. See Enable Secure Shell (SSH) Logins.

Ensure Ports Are Available

Verify that ports required by Vertica are not in use by running the following command as the root user and comparing it with the ports required, as shown below:
netstat -atupn

If you are using a Red Hat 7/CentOS 7 system, use the following command instead:

ss -atupn

Firewall Requirements

Vertica requires several ports to be open on the local network. Vertica does not recommend placing a firewall between nodes (all nodes should be behind a firewall), but if you must use a firewall between nodes, ensure the following ports are available:

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Service</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>TCP</td>
<td>sshd</td>
<td>Required by Administration Tools and the Management Console Cluster Installation wizard.</td>
</tr>
<tr>
<td>5433</td>
<td>TCP</td>
<td>Vertica</td>
<td>Vertica client (vsql, ODBC, JDBC, etc) port.</td>
</tr>
<tr>
<td>5434</td>
<td>TCP</td>
<td>Vertica</td>
<td>Intra- and inter-cluster communication. Vertica opens the Vertica client port +1 (5434 by default) for intra-cluster communication, such as during a plan. If the port +1 from the default client port is not available, then Vertica opens a random port for intra-cluster communication.</td>
</tr>
<tr>
<td>5433</td>
<td>UDP</td>
<td>Vertica</td>
<td>Vertica spread monitoring.</td>
</tr>
<tr>
<td>5444</td>
<td>TCP</td>
<td>Management Console</td>
<td>MC-to-node and node-to-node (agent) communications port. See Changing MC or Agent Ports.</td>
</tr>
<tr>
<td>5450</td>
<td>TCP</td>
<td>Management Console</td>
<td>Port used to connect to MC from a web browser and allows communication from nodes to the MC application/web server. See Connecting to Management Console.</td>
</tr>
<tr>
<td>4803</td>
<td>TCP</td>
<td>Spread</td>
<td>Client connections.</td>
</tr>
<tr>
<td>4803</td>
<td>UDP</td>
<td>Spread</td>
<td>Daemon to daemon connections.</td>
</tr>
<tr>
<td>4804</td>
<td>UDP</td>
<td>Spread</td>
<td>Daemon to daemon connections.</td>
</tr>
<tr>
<td>6543</td>
<td>UDP</td>
<td>Spread</td>
<td>Monitor to daemon connection.</td>
</tr>
</tbody>
</table>
Operating System Configuration Task Overview

This topic provides a high-level overview of the OS settings required for Vertica. Each item provides a link to additional details about the setting and detailed steps on making the configuration change. The installer tests for all of these settings and provides hints, warnings, and failures if the current configuration does not meet Vertica requirements.

Before You Install the Operating System

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Platforms</td>
<td>Verify that your servers meet the platform requirements described in <a href="#">Supported Platforms</a>. Unsupported operating systems are detected by the installer.</td>
</tr>
<tr>
<td>LVM</td>
<td>Vertica Analytic Database supports Linux Volume Manager (LVM) on all supported operating systems. For information on LVM requirements and restrictions, see the section, <a href="#">Vertica Support for LVM</a>.</td>
</tr>
<tr>
<td>Filesystem</td>
<td>The filesystem for the Vertica data and catalog directories must be formatted as ext4.</td>
</tr>
<tr>
<td>Swap Space</td>
<td>A 2GB swap partition is required. Partition the remaining disk space in a single partition under &quot;/&quot;.</td>
</tr>
<tr>
<td>Disk Block Size</td>
<td>The disk block size for the Vertica data and catalog directories should be 4096 bytes, the default for ext4 filesystems.</td>
</tr>
<tr>
<td>Memory</td>
<td>For more information on sizing your hardware, see the <a href="#">Vertica Hardware Planning Guide</a>.</td>
</tr>
</tbody>
</table>
Firewall Considerations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firewall/Ports</td>
<td>Firewalls, if present, must be configured so as not to interfere with Vertica.</td>
</tr>
</tbody>
</table>

General Operating System Configuration - Automatically Configured by Installer

These general OS settings are automatically made by the installer if they do not meet Vertica requirements. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nice Limits</td>
<td>The database administration user must be able to <code>nice</code> processes back to the default level of 0.</td>
</tr>
<tr>
<td>min_free_kbytes</td>
<td>The <code>vm.min_free_kbytes</code> setting in <code>/etc/sysctl.conf</code> must be configured sufficiently high. The specific value depends on your hardware configuration.</td>
</tr>
<tr>
<td>User Open Files Limit</td>
<td>The open file limit for the dbadmin user should be at least 1 file open per MB of RAM, 65536, or the amount of RAM in MB; whichever is greater.</td>
</tr>
<tr>
<td>System Open File Limits</td>
<td>The maximum number of files open on the system must not be less than at least the amount of memory in MB, but not less than 65536.</td>
</tr>
<tr>
<td>Pam Limits</td>
<td><code>/etc/pam.d/su</code> must contain the line: <code>session required pam_limits.so</code> This allows for the conveying of limits to commands run with the <code>su</code> command.</td>
</tr>
<tr>
<td>Address Space Limits</td>
<td>The address space limits (as setting) defined in <code>/etc/security/limits.conf</code> must be <code>unlimited</code> for the database administrator.</td>
</tr>
<tr>
<td>File Size Limits</td>
<td>The file sizelimits (<code>fsize</code> setting) defined in <code>/etc/security/limits.conf</code> must</td>
</tr>
</tbody>
</table>
General Operating System Configuration - Manual Configuration

The following general OS settings must be done manually.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Disk Readahead</strong></td>
<td>This disk readahead must be at least 2048, with a high of 8192. Set this high limit only with the help of Vertica support. The specific value depends on your hardware configuration.</td>
</tr>
<tr>
<td><strong>NTP Services</strong></td>
<td>The NTP daemon must be enabled and running, with the exception of Red Hat 7 and CentOS 7 systems.</td>
</tr>
<tr>
<td><strong>chrony</strong></td>
<td>For Red Hat 7 and CentOS 7 systems, chrony must be enabled and running.</td>
</tr>
<tr>
<td><strong>SELinux</strong></td>
<td>SELinux must be disabled or run in permissive mode.</td>
</tr>
<tr>
<td><strong>CPU Frequency Scaling</strong></td>
<td>Vertica recommends that you disable CPU Frequency Scaling.</td>
</tr>
<tr>
<td></td>
<td><strong>Important:</strong> Your systems may use significantly more energy when CPU frequency scaling is disabled.</td>
</tr>
<tr>
<td><strong>Transparent Hugepages</strong></td>
<td>For Red Hat 7 and CentOS 7 systems, Transparent Hugepages must be set to <em>always</em>.</td>
</tr>
<tr>
<td></td>
<td>For all other operating systems, Transparent Hugepages must be disabled or set to <em>madvise</em>.</td>
</tr>
<tr>
<td><strong>I/O Scheduler</strong></td>
<td>The I/O Scheduler for disks used by Vertica must be set to <em>deadline</em> or <em>noop</em>.</td>
</tr>
</tbody>
</table>

**User Process Limits**
The nproc setting defined in `/etc/security/limits.conf` must be 1024 or the amount of memory in MB, whichever is greater.

**Maximum Memory Maps**
The `vm.max_map_count` in `/etc/sysctl.conf` must be **65536** or the amount of memory in KB / 16, whichever is greater.
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Support Tools</strong></td>
<td>Several optional packages can be installed to assist Vertica support when troubleshooting your system.</td>
</tr>
</tbody>
</table>

**System User Requirements**

The following tasks pertain to the configuration of the system user required by Vertica.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Required Setting(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System User Requirements</strong></td>
<td>The installer automatically creates a user with the correct settings. If you specify a user with --dba-user, then the user must conform to the requirements for the Vertica system user.</td>
</tr>
<tr>
<td><strong>LANG Environment Settings</strong></td>
<td>The LANG environment variable must be set and valid for the database administration user.</td>
</tr>
<tr>
<td><strong>TZ Environment Settings</strong></td>
<td>The TZ environment variable must be set and valid for the database administration user.</td>
</tr>
</tbody>
</table>

**Operating System Prerequisites**

The topics in this section detail system settings that must be configured when you install the operating system. These settings cannot be easily changed after the operating system is installed.

**Supported Platforms**

The Vertica installer checks the type of operating system that is installed. If the operating system does not meet one of the supported operating systems (See Vertica Server and Vertica Management Console), or the operating system cannot be determined, then the installer halts.

The installer generates one of the following issue identifiers if it detects an unsupported operating system:

- [S0320] - Fedora OS is not supported.
- [S0321] - The version of Red Hat/CentOS is not supported.
[S0322] - The version of Ubuntu/Debian is not supported.

[S0323] - The operating system could not be determined. The unknown operating system is not supported because it does not match the list of supported operating systems.

[S0324] - The version of Red Hat is not supported.

**Filesystem Requirement**

Vertica requires that your Linux filesystem be ext4. All other filesystem types are unsupported. The installer reports this issue with the identifier S0160.

**Swap Space Requirements**

Vertica requires at least 2 GB swap partition regardless of the amount of RAM installed on your system. The installer reports this issue with identifier S0180.

For typical installations Vertica recommends that you partition your system with a 2GB primary partition for swap regardless of the amount of installed RAM. Larger swap space is acceptable, but unnecessary.

**Note:** Do not place a swap file on a disk containing the Vertica data files. If a host has only two disks (boot and data), put the swap file on the boot disk.

If you do not have at least a 2 GB swap partition then you may experience performance issues when running Vertica.

You typically define the swap partition when you install Linux. See your platform’s documentation for details on configuring the swap partition.

**Disk Block Size Requirements**

Vertica recommends that the disk block size be 4096 bytes, the default on ext4 filesystems. The installer reports this issue with the identifier S0165.

The disk block size is set when you format your file system. Changing the block size requires a re-format.
Memory Requirements

Vertica requires, at a minimum, 1GB of RAM per logical processor. The installer reports this issue with the identifier S0190.

However, for performance reasons, you typically require more RAM than the minimum. For more information on sizing your hardware, see the Vertica Hardware Planning Guide.

Firewall Considerations

Vertica requires multiple ports be open between nodes. You may use a firewall (IP Tables) on Redhat/CentOS and Ubuntu/Debian based systems. Note that firewall use is not supported on SuSE systems and that SuSE systems must disable the firewall. The installer reports issues found with your IP tables configuration with the identifiers N0010 for (systems that use IP Tables) and N011 (for SuSE systems).

The installer checks the IP tables configuration and issues a warning if there are any configured rules or chains. The installer does not detect if the configuration may conflict with Vertica. It is your responsibility to verify that your firewall allows traffic for Vertica as described in Ensure Ports Are Available.

Note: The installer does not check NAT entries in iptables.

You can modify your firewall to allow for Vertica network traffic, or you can disable the firewall if your network is secure. Note that firewalls are not supported for Vertica systems running on SuSE.

Important: You may encounter the N0010 issue even when the firewall is disabled. If this occurs, you can workaround this issue and install Vertica by ignoring installer WARN messages. To do this, install (or update) with a failure threshold of FAIL. For example, /opt/vertica/sbin/install_vertica --failure-threshold FAIL <other install options...>.

Red Hat 6 and CentOS 6 Systems

For details on how to configure iptables and allow specific ports to be open, see the platform-specific documentation for your platform:
To disable iptables, run the following command as root or sudo:

```bash
# service iptables save
# service iptables stop
# chkconfig iptables off
```

To disable iptables if you are using the ipv6 versions of iptables, run the following command as root or sudo:

```bash
# service ip6tables save
# service ip6tables stop
# chkconfig ip6tables off
```

**Red Hat 7 and CentOS 7 Systems:**

To disable the system firewall, run the following command as root or sudo:

```bash
# systemctl mask firewalld
# systemctl disable firewalld
# systemctl stop firewalld
```

**Ubuntu and Debian Systems**

For details on how to configure iptables and allow specific ports to be open, see the platform-specific documentation for your platform:

- **Debian:** https://wiki.debian.org/iptables
- **Ubuntu:** https://help.ubuntu.com/12.04/serverguide/firewall.html.

**Note:** Ubuntu uses the ufw program to manage iptables.

To disable iptables on Debian, run the following command as root or sudo:

```bash
/etc/init.d/iptables stop
update-rc.d -f iptables remove
```

To disable iptables on Ubuntu, run the following command:
SuSE Systems

The firewall must be disabled on SUSE systems. To disable the firewall on SuSE systems, run the following command:

```
ufw disable
```

Port Availability

The install_vertica script checks that required ports are open and available to Vertica. The installer reports any issues with identifier N0020.

The following table lists the ports required by Vertica.

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Service</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>TCP</td>
<td>sshd</td>
<td>Required by Administration Tools and the Management Console Cluster Installation wizard.</td>
</tr>
<tr>
<td>5433</td>
<td>TCP</td>
<td>Vertica</td>
<td>Vertica client (vsql, ODBC, JDBC, etc) port.</td>
</tr>
<tr>
<td>5434</td>
<td>TCP</td>
<td>Vertica</td>
<td>Intra- and inter-cluster communication. Vertica opens the Vertica client port +1 (5434 by default) for intra-cluster communication, such as during a plan. If the port +1 from the default client port is not available, then Vertica opens a random port for intra-cluster communication.</td>
</tr>
<tr>
<td>5433</td>
<td>UDP</td>
<td>Vertica</td>
<td>Vertica spread monitoring.</td>
</tr>
<tr>
<td>5444</td>
<td>TCP</td>
<td>Vertica</td>
<td>MC-to-node and node-to-node (agent) communications port. See Changing MC or Agent Ports.</td>
</tr>
<tr>
<td>5450</td>
<td>TCP</td>
<td>Vertica</td>
<td>Port used to connect to MC from a web browser and allows communication from nodes to the MC application/web server. See Connecting to Management Console.</td>
</tr>
<tr>
<td>Port</td>
<td>Protocol</td>
<td>Service</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>4803</td>
<td>TCP</td>
<td>Spread</td>
<td>Client connections.</td>
</tr>
<tr>
<td>4803</td>
<td>UDP</td>
<td>Spread</td>
<td>Daemon to daemon connections.</td>
</tr>
<tr>
<td>4804</td>
<td>UDP</td>
<td>Spread</td>
<td>Daemon to daemon connections.</td>
</tr>
<tr>
<td>6543</td>
<td>UDP</td>
<td>Spread</td>
<td>Monitor to daemon connection.</td>
</tr>
</tbody>
</table>

**General Operating System Configuration - Automatically Configured by the Installer**

These general Operating System settings are automatically made by the installer if they do not meet Vertica requirements. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

**sysctl**

During installation, Vertica attempts to automatically change various OS level settings. The installer may not change values on your system if they exceed the threshold required by the installer. You can prevent the installer from automatically making these configuration changes by using the `--no-system-configuration` parameter for the `install_vertica` script.

To permanently edit certain settings and prevent them from reverting on reboot, use `sysctl`. The `sysctl` settings relevant to the installation of Vertica include:

- `min_free_kbytes`
- `fs.file_max`
- `vm.max_map_count`

**Permanently Changing Settings with sysctl:**

1. As the root user, open the `/etc/sysctl.conf` file:

   ```sh
   # vi /etc/sysctl.conf
   ```
2. Enter a parameter and value:

```
parameter = value
```

For example, to set the parameter and value for `fs.file-max` to meet Vertica requirements, enter:

```
fs.file-max = 65536
```

3. Save your changes, and close the `/etc/sysctl.conf` file.

4. As the root user, reload the config file:

```
# sysctl -p
```

Identifying Settings Added by the Installer

You can see whether the installer has added a setting by opening the `/etc/sysctl.conf` file:

```
# vi /etc/sysctl.conf
```

If the installer has added a setting, the following line appears:

```
# The following 1 line added by Vertica tools. 2015-02-23 13:20:29
parameter = value
```

Nice Limits Configuration

The Vertica system user (dbadmin by default) must be able to raise and lower the priority of Vertica processes. To do this, the `nice` option in the `/etc/security/limits.conf` file must include an entry for the dbadmin user. The installer reports this issue with the identifier: `S0010`.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

**Note:** Vertica never raises priority above the default level of 0. However, Vertica does lower the priority of certain Vertica threads and needs to be able to raise the priority of these threads back up to the default level. This setting allows Vertica to raise the priorities back to the default level.
All Systems

To set the Nice Limit configuration for the dbadmin user, edit /etc/security/limits.conf and add the following line. Replace dbadmin with the name of your system user.

```
dbadmin - nice 0
```

min_free_kbytes Setting

This topic details how to update the min_free_kbytes setting so that it is within the range supported by Vertica. The installer reports this issue with the identifier: S0050 if the setting is too low, or S0051 if the setting is too high.

The vm.min_free_kbytes setting configures the page reclaim thresholds. When this number is increased the system starts reclaiming memory earlier, when its lowered it starts reclaiming memory later. The default min_free_kbytes is calculated at boot time based on the number of pages of physical RAM available on the system.

The setting must be the greater of:

- The default value configured by the system, or
- 4096, or
- determine the value from running the command below.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To manually set min_free_kbytes:

1. Determine the current/default setting with the following command:

```
/sbin/sysctl vm.min_free_kbytes
```
2. If the result of the previous command is No such file or directory or the default value is less than 4096, then run the command below:

```bash
memtot=`grep MemTotal /proc/meminfo | awk '{printf "%.0f",$2}'`
echo "scale=0;sqrt ($memtot*16)" | bc
```

3. Edit or add the current value of `vm.min_free_kbytes` in `/etc/sysctl.conf` with the value from the output of the previous command.

```bash
# The min_free_kbytes setting
vm.min_free_kbytes=5572
```

4. Run `sysctl -p` to apply the changes in `sysctl.conf` immediately.

Note: These steps will need to be replicated for each node in the cluster.

### User Max Open Files Limit

This topic details how to change the user max open-files limit setting to meet Vertica requirements. The installer reports this issue with the identifier: S0060.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

Vertica requires that the dbadmin user not be limited when opening files. The open file limit should be at least 1 file open per MB of RAM, 65536, or the amount of RAM in MB; whichever is greater. Vertica sets this to the minimum recommended value of 65536 or the amount of RAM in MB.

#### All Systems

The open file limit can be determined by running `ulimit -n` as the dbadmin user. For example:

```bash
dbadmin@localhost:$ ulimit -n
65536
```
To manually set the limit, edit `/etc/security/limits.conf` and edit/add the line for the `nofile` setting for the user you configured as the database admin (default `dbadmin`). The setting must be at least 65536.

```bash
dbadmin - nofile 65536
```

**Note:** There is also an open file limit on the system. See [System Max Open Files Limit](#).

## System Max Open Files Limit

This topic details how to modify the limit for the number of open files on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: `S0120`.

Vertica opens many files. Some platforms have global limits on the number of open files. The open file limit must be set sufficiently high so as not to interfere with database operations.

The recommended value is at least the amount of memory in MB, but not less than 65536.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

### All Systems

To manually set the open file limit:

1. Run `/sbin/sysctl fs.file-max` to determine the current limit.

2. If the limit is not `65536` or the amount of system memory in MB (whichever is higher), then edit or add `fs.file-max=max number of files to /etc/sysctl.conf`.

   ```bash
   # Controls the maximum number of open files
   fs.file-max=65536
   
   # Controls the maximum number of open files
   fs.file-max=65536
   
   # Controls the maximum number of open files
   fs.file-max=65536
   
   # Controls the maximum number of open files
   fs.file-max=65536
   
   # Controls the maximum number of open files
   fs.file-max=65536
   
   # Controls the maximum number of open files
   fs.file-max=65536
   ```

3. Run `sysctl -p` to apply the changes in `sysctl.conf` immediately.

**Note:** These steps will need to be replicated for each node in the cluster.
Pam Limits

This topic details how to enable the "su" pam_limits.so module required by Vertica. The installer reports issues with the setting with the identifier: S0070.

On some systems the pam module called pam_limits.so is not set in the file /etc/pam.d/su. When it is not set, it prevents the conveying of limits (such as open file descriptors) to any command started with su -.

In particular, the Vertica init script would fail to start Vertica because it calls the Administration Tools to start a database with the su - command. This problem was first noticed on Debian systems, but the configuration could be missing on other Linux distributions. See the pam_limits man page for more details.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To manually configure this setting, append the following line to the /etc/pam.d/su file:

```
session required pam_limits.so
```

See the pam_limits man page for more details: man pam_limits.

pid_max Setting

This topic explains how to change pid_max to a supported value. The value of pid_max should be

```
pid_max = num_user_proc + 2**15 = num_user_proc + 32768
```

where num_user_proc is the size of memory in megabytes.

The minimum value for pid_max is 524288.

If your pid_max value is too low, the installer reports this problem and indicates the minimum value.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-
system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

All Systems

To change the pid_max value:

```
# sysctl -w kernel.pid_max=524288
```

**User Address Space Limits**

This topic details how to modify the Linux address space limit for the dbadmin user so that it meets Vertica requirements. The address space setting controls the maximum number of threads and processes for each user. If this setting does not meet the requirements then the installer reports this issue with the identifier: S0090.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The address space available to the dbadmin user must not be reduced via user limits and must be set to unlimited.

All Systems

To manually set the address space limit:

1. Run `ulimit -v` as the dbadmin user to determine the current limit.

2. If the limit is not unlimited, then add the following line to `/etc/security/limits.conf`. Replace `dbadmin` with your database admin user

   ```
dbadmin -v unlimited
   ```

**User File Size Limit**

This topic details how to modify the file size limit for files on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: S0100.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-
system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The file size limit for the dbadmin user must not be reduced via user limits and must be set to unlimited.

All Systems

To manually set the file size limit:

1. Run `ulimit -f` as the dbadmin user to determine the current limit.
2. If the limit is not unlimited, then edit/add the following line to `/etc/security/limits.conf`. Replace `dbadmin` with your database admin user.

```
    dbadmin    fsize    unlimited
```

User Process Limit

This topic details how to change the user process limit so that it meets Vertica requirements. The installer reports this issue with the identifier: S0110.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the `--no-system-configuration` argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

The user process limit must be high enough to allow for the many threads opened by Vertica. The recommended limit is the amount of RAM in MB and must be at least 1024.

All Systems

To manually set the user process limit:

1. Run `ulimit -u` as the dbadmin user to determine the current limit.
2. If the limit is not the amount of memory in MB on the server, then edit/add the following line to `/etc/security/limits.conf`. Replace `4096` with the amount of system memory, in MB, on the server.

```
    dbadmin    nproc    4096
```
Maximum Memory Maps Configuration

This topic details how to modify the limit for the number memory maps a process can have on your system so that it meets Vertica requirements. The installer reports this issue with the identifier: S0130.

The installer automatically configures the correct setting if the default value does not meet system requirements. If there is an issue setting this value, or you have used the --no-system-configuration argument to the installer and the current setting is incorrect, then the installer reports this as an issue.

Vertica uses a lot of memory while processing and can approach the default limit for memory maps per process.

The recommended value is at least the amount of memory on the system in KB / 16, but not less than 65536.

All Systems

To manually set the memory map limit:

1. Run /sbin/sysctl vm.max_map_count to determine the current limit.

2. If the limit is not 65536 or the amount of system memory in KB / 16 (whichever is higher), then edit/add the following line to /etc/sysctl.conf. Replace 65536 with the value for your system.

   # The following 1 line added by Vertica tools. 2014-03-07 13:20:31
   vm.max_map_count=65536

3. Run sysctl -p to apply the changes in sysctl.conf immediately.

   Note: These steps will need to be replicated for each node in the cluster.

General Operating System Configuration - Manual Configuration

The following general Operating System settings must be done manually.
Manually Configuring Operating System Settings

Vertica requires that you manually configure some general operating system settings. Vertica recommends that you configure these settings in the /etc/rc.local script to prevent them from reverting on reboot. The /etc/rc.local startup script contains scripts and commands that run each time the system is booted.

If you are using Red Hat 7.0 or CentOS 7.0 or higher, you must make sure the tuning system service will not start upon reboot.

Run the following command as sudo or root:

```
$ chkconfig tuned off
```

Turning off tuning prevents monitoring of your OS and any tuning of your OS based on this monitoring. Tuning also enables THP silently which may cause issues in other areas, for example read ahead.

**Note:** SUSE systems use the /etc/init.d/after.local file rather than the etc/rc.local file. For purposes of using Vertica, the functionality of both files is the same.

Settings to Configure Manually

The /etc/rc.local settings relevant to the installation of Vertica include:

- Disk Readahead
- I/O Scheduling
- Enabling or Disabling Transparent Hugepages

Permanently Changing Settings with /etc/rc.local

1. As the root user, open /etc/rc.local:

   ```
   # vi /etc/rc.local
   ```

2. Enter a script or command. For example, to set the transparent hugepages setting to meet Vertica requirements, enter the following:

   ```
   echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
   ```
Important: On some Ubuntu/Debian systems, the last line in /etc/rc.local must be "exit 0". Any additions to /etc/rc.local must come before "exit 0".

3. Save your changes, and close the /etc/rc.local file.

4. If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```bash
$ chmod +x /etc/rc.d/rc.local
```

On the next reboot, the command runs during startup. You can also run the command manually, as the root user, if you want it to take effect immediately.

**Check for Swappiness**

The swappiness kernel parameter defines the amount, and how often, the kernel copies RAM contents to a swap space. Vertica recommends a value of 1. The installer reports any swappiness issues with identifier S0112.

To set the swappiness value add or update the following in /etc/sysctl.conf:

```bash
vm.swappiness = 1
```

This also ensures that the value persists after a reboot.

You can check the swappiness value as follows:

```bash
cat /proc/sys/vm/swappiness
```

If necessary, you change the swappiness value at runtime by logging in as root and running the following:

```bash
echo 1 > /proc/sys/vm/swappiness
```

**Disk Readahead**

This topic details how to change Disk Readahead to a supported value. Vertica requires that Disk Readahead be set to at least 2048. The installer reports this issue with the identifier: S0020.

**Note:**
These commands must be executed with root privileges and assumes the blockdev program is in /sbin.

The blockdev program operates on whole devices, and not individual partitions. You cannot set the readahead value to different settings on the same device. If you run blockdev against a partition, for example: /dev/sda1, then the setting is still applied to the entire /dev/sda device. For instance, running /sbin/blockdev --setra 2048 /dev/sda1 also causes /dev/sda2 through /dev/sdaN to use a readahead value of 2048.

RedHat/CentOS and SuSE Based Systems

For each drive in the Vertica system, Vertica recommends that you set the readahead value to at least 2048 for most deployments. The command immediately changes the readahead value for the specified disk. The second line adds the command to /etc/rc.local so that the setting is applied each time the system is booted. Note that some deployments may require a higher value and the setting can be set as high as 8192, under guidance of support.

```
Note: For systems that do not support /etc/rc.local, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses /etc/init.d/after.local.
```

```
/sbin/blockdev --setra 2048 /dev/sda
echo '/sbin/blockdev --setra 2048 /dev/sda' >> /etc/rc.local
```

If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

Ubuntu and Debian Systems

For each drive in the Vertica system, set the readahead value to 2048. Run the command once in your shell, then add the command to /etc/rc.local so that the setting is applied each time the system is booted. Note that on Ubuntu systems, the last line in rc.local must be "exit 0". So you must manually add the following line to etc/rc.local before the last line with exit 0.

```
Note: For systems that do not support /etc/rc.local, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses /etc/init.d/after.local.
```

```
```
Enabling Network Time Protocol (NTP)

The network time protocol (NTP) daemon must be running on all of the hosts in the cluster so that their clocks are synchronized. The spread daemon relies on all of the nodes to have their clocks synchronized for timing purposes. If your nodes do not have NTP running, the installation can fail with a spread configuration error or other errors.

Note: Different Linux distributions refer to the NTP daemon in different ways. For example, SUSE and Debian/Ubuntu refer to it as ntp, while CentOS and Red Hat refer to it as ntpd. If the following commands produce errors, try using ntp in place of ntpd.

Verify That NTP Is Running

To verify that your hosts are configured to run the NTP daemon on startup, enter the following command:

```
$ chkconfig --list ntpd
```

Debian and Ubuntu do not support `chkconfig`, but they do offer an optional package. You can install this package with the command `sudo apt-get install sysv-rc-conf`. To verify that your hosts are configured to run the NTP daemon on startup with the `sysv-rc-conf` utility, enter the following command:

```
$ sysv-rc-conf --list ntpd
```

The `chkconfig` command can produce an error similar to `ntpd: unknown service`. If you get this error, verify that your Linux distribution refers to the NTP daemon as `ntpd` rather than `ntp`. If it does not, you need to install the NTP daemon package before you can configure it. Consult your Linux documentation for instructions on how to locate and install packages.

If the NTP daemon is installed, your output should resemble the following:

```
ntp 0:off 1:off 2:on 3:on 4:off 5:on 6:off
```

The output indicates the runlevels where the daemon runs. Verify that the current runlevel of the system (usually 3 or 5) has the NTP daemon set to on. If you do not know the current runlevel, you can find it using the `runlevel` command:

```
$ runlevel
N 3
```
Configure NTP for Red Hat 6/CentOS 6 and SLES

If your system is based on Red Hat 6/CentOS 6 or SUSE Linux Enterprise Server, use the service and chkconfig utilities to start NTP and have it start at startup.

```
/sbin/service ntpd restart
/sbin/chkconfig ntpd on
```

- **Red Hat 6/CentOS 6**—NTP uses the default time servers at ntp.org. You can change the default NTP servers by editing `/etc/ntpd.conf`.

- **SLES**—By default, no time servers are configured. You must edit `/etc/ntpd.conf` after the install completes and add time servers.

Configure NTP for Ubuntu and Debian

By default, the **NTP daemon** is not installed on some Ubuntu and Debian systems. First, install NTP, and then start the NTP process. You can change the default NTP servers by editing `/etc/ntpd.conf` as shown:

```
sudo apt-get install ntp
sudo /etc/init.d/ntp reload
```

Verify That NTP Is Operating Correctly

To verify that the Network Time Protocol Daemon (NTPD) is operating correctly, issue the following command on all nodes in the cluster.

**For Red Hat 6/CentOS 6 and SLES:**

```
/usr/sbin/ntpq -c rv | grep stratum
```

**For Ubuntu and Debian:**

```
ntpq -c rv | grep stratum
```

A stratum level of 16 indicates that NTP is not synchronizing correctly.

If a stratum level of 16 is detected, wait 15 minutes and issue the command again. It may take this long for the NTP server to stabilize.

If NTP continues to detect a stratum level of 16, verify that the NTP port (UDP Port 123) is open on all firewalls between the cluster and the remote machine to which you are attempting to synchronize.
Red Hat Documentation Related to NTP

The preceding links were current as of the last publication of the Vertica documentation and could change between releases.

- http://kbase.redhat.com/faq/docs/DOC-6731
- http://kbase.redhat.com/faq/docs/DOC-6902
- http://kbase.redhat.com/faq/docs/DOC-6991

Enabling chrony or ntpd for Red Hat 7/CentOS 7 Systems

Before you can install Vertica, you must enable one of the following on your system for clock synchronization:

- chrony
- NTPD

You must enable and activate the Network Time Protocol (NTP) before installation. Otherwise, the installer reports this issue with the identifier S0030.

For information on installing and using chrony, see the information below. For information on NTPD see Enabling Network Time Protocol (NTP).

Install chrony

The chrony suite consists of:

- chronyd - the daemon for clock synchronization.
- chronyc - the command-line utility for configuring chronyd.

chrony is installed by default on some versions of Red Hat/CentOS 7. However, if chrony is not installed on your system, you must download it. To download chrony, run the following command as sudo or root:

```bash
# yum install chrony
```

Verify That chrony Is Running

To view the status of the chronyd daemon, run the following command:
If `chrony` is running, an output similar to the following appears:

```
chronyd.service - NTP client/server
 Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled)
 Active: active (running) since Mon 2015-07-06 16:29:54 EDT; 15s ago
 Main PID: 2530 (chronyd)
 CGroup: /system.slice/chronyd.service
     ââ2530 /usr/sbin/chronyd -u chrony
```

If `chrony` is not running, execute the following command as `sudo` or `root`. This command also causes `chrony` to run at boot time:

```
# systemctl enable chronyd
```

Verify That `chrony` Is Operating Correctly

To verify that the `chrony` daemon is operating correctly, issue the following command on all nodes in the cluster:

```
$ chronyc tracking
```

An output similar to the following appears:

```
Reference ID      : 198.247.63.98 (time01.website.org)
Stratum           : 3
Ref time (UTC)    : Thu Jul  9 14:58:01 2015
System time       : 0.000035685 seconds slow of NTP time
Last offset       : -0.000151098 seconds
RMS offset        : 0.000279871 seconds
Frequency         : 2.085 ppm slow
Residual freq     : -0.013 ppm
Skew              : 0.185 ppm
Root delay        : 0.042370 seconds
Root dispersion   : 0.022658 seconds
Update interval   : 1031.0 seconds
Leap status       : Normal
```

A stratum level of 16 indicates that `chrony` is not synchronizing correctly. If `chrony` continues to detect a stratum level of 16, verify that the UDP port 323 is open. This port must be open on all firewalls between the cluster and the remote machine to which you are attempting to synchronize.

Red Hat Documentation Related to `chrony`

These links to Red Hat documentation were current as of the last publication of the Vertica documentation. Be aware that they could change between releases:
- Configuring NTP Using the chrony Suite
- Using chrony

Cron Required for Scheduled Jobs

Admintools uses the Linux `cron` package to schedule jobs that regularly rotate the database logs. Without this package installed, the database logs will never be rotated. The lack of rotation can lead to a significant consumption of storage for logs. On busy clusters, Vertica can produce hundreds of gigabytes of logs per day.

`cron` is installed by default on most Linux distributions, but it may not be present on some SUSE 12 systems.

To install `cron`, run this command:

```
$ sudo zypper install cron
```

SELinux Configuration

Vertica does not support SELinux except when SELinux is running in permissive mode. If it detects that SELinux is installed and the mode cannot be determined the installer reports this issue with the identifier: S0080. If the mode can be determined, and the mode is not permissive, then the issue is reported with the identifier: S0081.

Red Hat and SUSE Systems

You can either disable SELinux or change it to use permissive mode.

To disable SELinux:

1. Edit `/etc/selinux/config` and change setting for SELinux to disabled (SELINUX=disabled). This disables SELinux at boot time.

2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/etc/selinux/config` and change setting for SELINUX to permissive (SELINUX=Permissive).
2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

Ubuntu and Debian Systems

You can either disable SELinux or change it to use permissive mode.

To disable SELinux:

1. Edit `/etc/selinux/config` and change setting for SELinux to disabled (SELINUX=disabled). This disables SELinux at boot time.

2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/etc/selinux/config` and change setting for SELinux to permissive (SELINUX=Permissive).

2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

CPU Frequency Scaling

This topic details the various CPU frequency scaling methods supported by Vertica. In general, if you do not require CPU frequency scaling, then disable it so as not to impact system performance.

**Important:** Your systems may use significantly more energy when frequency scaling is disabled.

The installer allows CPU frequency scaling to be enabled when the `cpufreq` scaling governor is set to **performance**. If the cpu scaling governor is set to **ondemand**, and `ignore_nice_load` is 1 (true), then the installer fails with the error S0140. If the cpu scaling governor is set to **ondemand** and `ignore_nice_load` is 0 (false), then the installer warns with the identifier S0141.

CPU frequency scaling is a hardware and software feature that helps computers conserve energy by slowing the processor when the system load is low, and speeding it up again when the system load increases. This feature can impact system performance, since raising the CPU frequency in response to higher system load does not occur instantly. Always disable this feature on the Vertica database hosts to prevent it from interfering with performance.
You disable CPU scaling in your host's system BIOS. There may be multiple settings in your host's BIOS that you need to adjust in order to completely disable CPU frequency scaling. Consult your host hardware's documentation for details on entering the system BIOS and disabling CPU frequency scaling.

If you cannot disable CPU scaling through the system BIOS, you can limit the impact of CPU scaling by disabling the scaling through the Linux kernel or setting the CPU frequency governor to always run the CPU at full speed.

**Caution:** This method is not reliable, as some hardware platforms may ignore the kernel settings. The only reliable method is to disable CPU scaling in BIOS.

The method you use to disable frequency depends on the CPU scaling method being used in the Linux kernel. See your Linux distribution's documentation for instructions on disabling scaling in the kernel or changing the CPU governor.

### Enabling or Disabling Transparent Hugepages

You can modify transparent hugepages so that the configuration meets Vertica requirements.

- For Red Hat 7/CentOS 7 systems, you must enable transparent hugepages. The installer reports this issue with the identifier: S0312.

- For all other systems, you must disable transparent hugepages or set them to `madvise`. The installer reports this issue with the identifier: S0310.

#### Disable Transparent Hugepages on Red Hat 6/CentOS 6 Systems

**Important:** If you are using Red Hat 7/CentOS 7, you must enable, rather than disable transparent hugepages. See: [Enable Transparent Hugepages on Red Hat 7/CentOS 7 Systems](#).

Determine if transparent hugepages is enabled. To do so, run the following command.

```
cat /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

The setting returned in brackets is your current setting.

If you are not using `madvise` or never as your transparent hugepage setting, then you can disable transparent hugepages in one of two ways:
Edit your boot loader (for example /etc/grub.conf). Typically, you add the following to the end of the kernel line. However, consult the documentation for your system before editing your boot loader configuration.

```bash
transparent_hugepage=never
```

Edit /etc/rc.local and add the following script.

```bash
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
fi
```

For systems that do not support /etc/rc.local, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses /etc/init.d/after.local.

Regardless of which approach you choose, you must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

```bash
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

Enable Transparent Hugepages on Red Hat 7/CentOS 7 Systems

Determine if transparent hugepages is enabled. To do so, run the following command.

```bash
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

The setting returned in brackets is your current setting.

For systems that do not support /etc/rc.local, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses /etc/init.d/after.local.

You can enable transparent hugepages by editing /etc/rc.local and adding the following script:

```bash
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
echo always > /sys/kernel/mm/transparent_hugepage/enabled
fi
```

You must reboot your system for the setting to take effect, or, as root, run the following echo line to proceed with the install without rebooting:

```bash
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```
If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

**Disable Transparent Hugepages on Other Systems**

**Note:** SuSE did not offer transparent hugepage support in its initial 11.0 release. However, subsequent SuSE service packs do include support for transparent hugepages.

To determine if transparent hugepages is enabled, run the following command.

```
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

The setting returned in brackets is your current setting. Depending on your platform OS, the `madvise` setting may not be displayed.

You can disable transparent hugepages one of two ways:

- **Edit your boot loader** (for example `/etc/grub.conf`). Typically, you add the following to the end of the kernel line. However, consult the documentation for your system before editing your bootloader configuration.

  ```
  transparent_hugepage=never
  ```

- **Edit `/etc/rc.local** (on systems that support `rc.local`) and add the following script.

  ```
  if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/enabled
  fi
  ```

For systems that do not support `/etc/rc.local`, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses `/etc/init.d/after.local`.

Regardless of which approach you choose, you must reboot your system for the setting to take effect, or run the following two echo lines to proceed with the install without rebooting:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```
Disabling Defrag for Red Hat and CentOS Systems

On all Red Hat and CentOS systems, you must disable the defrag utility to meet Vertica configuration requirements. The steps necessary to disable defrag on Red Hat 6/CentOS 6 systems differ from those used to disable defrag on Red Hat 7/CentOS 7 systems.

Disable Defrag on Red Hat 6/CentOS 6 Systems

1. Determine if defrag is enabled by running the following command:

   ```bash
   cat /sys/kernel/mm/redhat_transparent_hugepage/defrag
   [always] madvise never
   ```

   The setting returned in brackets is your current setting. If you are not using `madvise` or `never` as your defrag setting, then you must disable defrag.

2. Edit `/etc/rc.local`, and add the following script:

   ```bash
   if test -f /sys/kernel/mm/redhat_transparent_hugepage/defrag-enabled; then
     echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
   fi
   ```

   You must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

   ```bash
   # echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
   ```

Disable Defrag on Red Hat 7/CentOS 7 Systems

1. Determine if defrag is enabled by running the following command:

   ```bash
   cat /sys/kernel/mm/transparent_hugepage/defrag
   [always] madvise never
   ```

   The setting returned in brackets is your current setting. If you are not using `madvise` or `never` as your defrag setting, then you must disable defrag.

2. Edit `/etc/rc.local`, and add the following script:

   ```bash
   if test -f /sys/kernel/mm/transparent_hugepage/defrag-enabled; then
     echo never > /sys/kernel/mm/transparent_hugepage/defrag
   fi
   ```
You must reboot your system for the setting to take effect, or run the following echo line to proceed with the install without rebooting:

```
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

3. If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

### I/O Scheduling

This topic details how to change I/O Scheduling to a supported scheduler. Vertica requires that I/O Scheduling be set to deadline or noop. The installer checks what scheduler the system is using, reporting an unsupported scheduler issue with identifier: S0150. If the installer cannot detect the type of scheduler in use (typically if your system is using a RAID array), it reports that issue with identifier: S0151.

If your system is not using a RAID array, then complete the following steps to change your system to a supported I/O Scheduler. If you are using a RAID array, then consult your RAID vendor documentation for the best performing scheduler for your hardware.

#### Configure the I/O Scheduler

The Linux kernel can use several different I/O schedulers to prioritize disk input and output. Most Linux distributions use the Completely Fair Queuing (CFQ) scheme by default, which gives input and output requests equal priority. This scheduler is efficient on systems running multiple tasks that need equal access to I/O resources. However, it can create a bottleneck when used on Vertica drives containing the catalog and data directories, because it gives write requests equal priority to read requests, and its per-process I/O queues can penalize processes making more requests than other processes.

Instead of the CFQ scheduler, configure your hosts to use either the Deadline or NOOP I/O scheduler for the drives containing the catalog and data directories:

- The Deadline scheduler gives priority to read requests over write requests. It also imposes a deadline on all requests. After reaching the deadline, such requests gain priority over all other requests. This scheduling method helps prevent processes from becoming starved for I/O access. The Deadline scheduler is best used on physical media drives (disks using spinning platters), since it attempts to group requests for adjacent sectors on a disk,
lowering the time the drive spends seeking.

- The NOOP scheduler uses a simple FIFO approach, placing all input and output requests into a single queue. This scheduler is best used on solid state drives (SSDs). Because SSDs do not have a physical read head, no performance penalty exists when accessing non-adjacent sectors.

Failure to use one of these schedulers for the Vertica drives containing the catalog and data directories can result in slower database performance. Other drives on the system (such as the drive containing swap space, log files, or the Linux system files) can still use the default CFQ scheduler (although you should always use the NOOP scheduler for SSDs).

There are two ways for you to set the scheduler used by your disk devices:

1. Write the name of the scheduler to a file in the /sys directory.

   --or--

2. Use a kernel boot parameter.

Configure the I/O Scheduler - Changing the Scheduler Through the /sys Directory

You can view and change the scheduler Linux uses for I/O requests to a single drive using a virtual file under the /sys directory. The name of the file that controls the scheduler a block device uses is:

```
/sys/block/deviceName/queue/scheduler
```

Where `deviceName` is the name of the disk device, such as `sda` or `cciss\!c0d1` (the first disk on an OpenText RAID array). Viewing the contents of this file shows you all of the possible settings for the scheduler. The currently-selected scheduler is surrounded by square brackets:

```
# cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

To change the scheduler, write the name of the scheduler you want the device to use to its scheduler file. You must have root privileges to write to this file. For example, to set the sda drive to use the deadline scheduler, run the following command as root:

```
# echo deadline > /sys/block/sda/queue/scheduler
# cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

Changing the scheduler immediately affects the I/O requests for the device. The Linux kernel starts using the new scheduler for all of the drive's input and output requests.
Note: While tests show that changing the scheduler settings while Vertica is running does not cause problems, Vertica recommends shutting down. Before changing the I/O schedule, or making any other changes to the system configuration, consider shutting down any running database.

Changes to the I/O scheduler made through the /sys directory only last until the system is rebooted, so you need to add the commands that change the I/O scheduler to a startup script (such as those stored in /etc/init.d, or though a command in /etc/rc.local). You also need to use a separate command for each drive on the system whose scheduler you want to change.

For example, to make the configuration take effect immediately and add it to rc.local so it is used on subsequent reboots.

Note: For systems that do not support /etc/rc.local, use the equivalent startup script that is run after the destination runlevel has been reached. For example SuSE uses /etc/init.d/after.local.

```
echo deadline > /sys/block/sda/queue/scheduler
echo 'echo deadline > /sys/block/sda/queue/scheduler' >> /etc/rc.local
```

Note: On some Ubuntu/Debian systems, the last line in rc.local must be "exit 0". So you must manually add the following line to etc/rc.local before the last line with exit 0.

You may prefer to use this method of setting the I/O scheduler over using a boot parameter if your system has a mix of solid-state and physical media drives, or has many drives that do not store Vertica catalog and data directories.

If you are using Red Hat 7.0 or CentOS 7.0 or higher, run the following command as root or sudo:

```
$ chmod +x /etc/rc.d/rc.local
```

Configure the I/O Scheduler - Changing the Scheduler with a Boot Parameter

Use the elevator kernel boot parameter to change the default scheduler used by all disks on your system. This is the best method to use if most or all of the drives on your hosts are of the same type (physical media or SSD) and will contain catalog or data files. You can also use the boot parameter to change the default to the scheduler the majority of the drives on the system need, then use the /sys files to change individual drives to another I/O scheduler. The format of the elevator boot parameter is:

```
elevator=schedulerName
```
Where *schedulerName* is deadline, noop, or cfq. You set the boot parameter using your bootloader (grub or grub2 on most recent Linux distributions). See your distribution's documentation for details on how to add a kernel boot parameter.

**Support Tools**

Vertica suggests that the following tools are installed so support can assist in troubleshooting your system if any issues arise:

- pstack (or gstack) package. Identified by issue S0040 when not installed.
  - On Red Hat 7 and CentOS 7 systems, the pstack package is installed as part of the gdb package.

- mcelog package. Identified by issue S0041 when not installed.

- sysstat package. Identified by issue S0045 when not installed.

**Red Hat 6 and CentOS 6 Systems**

To install the required tools on Red Hat 6 and CentOS 6 systems, run the following commands as sudo or root:

```
yum install pstack
yum install mcelog
yum install sysstat
```

**Red Hat 7 and CentOS 7 Systems**

To install the required tools on Red Hat 7/CentOS 7 systems, run the following commands as sudo or root:

```
yum install gdb
yum install mcelog
yum install sysstat
```

**Ubuntu and Debian Systems**

To install the required tools on Ubuntu and Debian systems, run the following commands as sudo or root:

```
apt-get install pstack
apt-get install mcelog
apt-get install sysstat
```
SuSE Systems

To install the required tools on SuSE systems, run the following commands as sudo or root.

```
zypper install sysstat
zypper install mcelog
```

There is no individual SuSE package for pstack/gstack. However, the gdb package contains gstack, so you could optionally install gdb instead, or build pstack/gstack from source. To install the gdb package:

```
zypper install gdb
```

System User Configuration

The following tasks pertain to the configuration of the system user required by Vertica.

System User Requirements

Vertica has specific requirements for the system user that runs and manages Vertica. If you specify a user during install, but the user does not exist, then the installer reports this issue with the identifier: S0200.

System User Requirement Details

Vertica requires a system user to own database files and run database processes and administration scripts. By default, the install script automatically configures and creates this user for you with the username dbadmin. See About Linux Users Created by Vertica and Their Privileges for details on the default user created by the install script. If you decide to manually create your own system user, then you must create the user before you run the install script. If you manually create the user:

**Note:** Instances of dbadmin and verticadba are placeholders for the names you choose if you do not use the default values.
• the user must have the same username and password on all nodes

• the user must use the BASH shell as the user's default shell. If not, then the installer reports this issue with identifier [S0240].

• the user must be in the verticadba group (for example: usermod -a -G verticadba userNameHere). If not, the installer reports this issue with identifier [S0220].

  Note: You must create a verticadba group on all nodes. If you do not, then the installer reports the issue with identifier [S0210].

• the user's login group must be either verticadba or a group with the same name as the user (for example, the home group for dbadmin is dbadmin). You can check the groups for a user with the id command. For example: id dbadmin. The "gid" group is the user's primary group. If this is not configured correctly then the installer reports this issue with the identifier [S0230]. Vertica recommends that you use verticadba as the user's primary login group. For example: usermod -g verticadba userNameHere. If the user's primary group is not verticadba as suggested, then the installer reports this with HINT [S0231].

• the user must have a home directory. If not, then the installer reports this issue with identifier [S0260].

• the user’s home directory must be owned by the user. If not, then the installer reports the issue with identifier [S0270].

• the system must be aware of the user's home directory (you can set it with the usermod command: usermod -m -d /path/to/new/home/dir userNameHere). If this is not configured correctly then the installer reports the issue with [S0250].

• the user's home directory must be owned by the dbadmin's primary group (use the chown and chgrp commands if necessary). If this is not configured correctly, then the installer reports the issue with identifier [S0280].

• the user’s home directory should have secure permissions. Specifically, it should not be writable by anyone or by the group. Ideally the permissions should be, when viewing with ls, "- - -" (nothing), or "r-x" (read and execute). If this is not configured as suggested then the installer reports this with HINT [S0290].
TZ Environment Variable

This topic details how to set or change the TZ environment variable and update your tzdata package. If this variable is not set, then the installer reports this issue with the identifier: S0305.

Before installing Vertica, update the tzdata package for your system and set the default time zone for your database administrator account by specifying the TZ environmental variable. If your database administrator is being created by the install_vertica script, then set the TZ variable after you have installed Vertica.

Update tzdata Package

The tzdata package is a public-domain time zone database that is pre-installed on most Linux systems. The tzdata package is updated periodically for time-zone changes across the world. OpenText recommends that you update to the latest tzdata package before installing or updating Vertica.

Update your tzdata package with the following command:

- RedHat based systems: `yum update tzdata`
- Debian and Ubuntu systems: `apt-get install tzdata`

Setting the Default Time Zone

When a client receives the result set of a SQL query, all rows contain data adjusted, if necessary, to the same time zone. That time zone is the default time zone of the initiator node unless the client explicitly overrides it using the SQL SET TIME ZONE command described in the SQL Reference Manual. The default time zone of any node is controlled by the TZ environment variable. If TZ is undefined, the operating system time zone.

**Important:** The TZ variable must be set to the same value on all nodes in the cluster.

If your operating system timezone is not set to the desired time zone of the database then make sure that the Linux environment variable TZ is set to the desired value on all cluster hosts.

The installer returns a warning if the TZ variable is not set. If your operating system timezone is appropriate for your database, then the operating system timezone is used and the warning can be safely ignored.
Setting the Time Zone on a Host

Important: If you explicitly set the TZ environment variable at a command line before you start the Administration Tools, the current setting will not take effect. The Administration Tools uses SSH to start copies on the other nodes, so each time SSH is used, the TZ variable for the startup command is reset. TZ must be set in the .profile or .bashrc files on all nodes in the cluster to take affect properly.

You can set the time zone several different ways, depending on the Linux distribution or the system administrator’s preferences.

- To set the system time zone on Red Hat and SUSE Linux systems, edit:

  ```
  /etc/sysconfig/clock
  ```

- To set the TZ variable, edit /etc/profile, or /home/dbadmin/.bashrc or /home/dbadmin/.bash_profile and add the following line (for example, for the US Eastern Time Zone):

  ```
  export TZ="America/New_York"
  ```

For details on which timezone names are recognized by Vertica, see the appendix: Using Time Zones With Vertica.

LANG Environment Variable Settings

This topic details how to set or change the LANG environment variable. The LANG environment variable controls the locale of the host. If this variable is not set, then the installer reports this issue with the identifier: S0300. If this variable is not set to a valid value, then the installer reports this issue with the identifier: S0301.

Set the Host Locale

Each host has a system setting for the Linux environment variable LANG. LANG determines the locale category for native language, local customs, and coded character set in the absence of the LC_ALL and other LC_ environment variables. LANG can be used by applications to determine which language to use for error messages and instructions, collating sequences, date formats, and so forth.
To change the LANG setting for the database administrator, edit, /etc/profile, or /dbadmin/.bashrc or /home/dbadmin/.bash_profile on all cluster hosts and set the environment variable; for example:

```
export LANG=en_US.UTF8
```

The LANG setting controls the following in Vertica:

- OS-level errors and warnings, for example, "file not found" during COPY operations.
- Some formatting functions, such as TO_CHAR and TO_NUMBER. See also Template Patterns for Numeric Formatting.

The LANG setting does not control the following:

- Vertica specific error and warning messages. These are always in English at this time.
- Collation of results returned by SQL issued to Vertica. This must be done using a database parameter instead. See Implement Locales for International Data Sets section in the Administrator’s Guide for details.

Note: If the LC_ALL environment variable is set, it supersedes the setting of LANG.

Package Dependencies

For successful Vertica installation, you must first install three packages on all nodes in your cluster before installing the database platform.

The required packages are:

- openssh—Required for Administration Tools connectivity between nodes.
- which—Required for Vertica operating system integration and for validating installations.
- dialog—Required for interactivity with Administration Tools.

Installing the Required Packages

The procedure you follow to install the required packages depends on the operating system on which your node or cluster is running. See your operating system's documentation for detailed information on installing packages.
For CentOS/Red Hat Systems—Typically, you manage packages on Red Hat and CentOS systems using the yum utility.

Run the following yum commands to install each of the package dependencies. The yum utility guides you through the installation:

```
# yum install openssh
# yum install which
# yum install dialog
```

For Debian/Ubuntu Systems—Typically, you use the apt-get utility to manage packages on Debian and Ubuntu systems.

Run the following apt-get commands to install each of the package dependencies. The apt-get utility guides you through the installation:

```
# apt-get install openssh
# apt-get install which
# apt-get install dialog
```
Installing Vertica

There are different paths you can take when installing Vertica. You can:

- Install Vertica on one or more hosts using the command line, and not use the Management Console.
- Install the Management Console, and from the Management Console install Vertica on one or more hosts by using the Management Console cluster creation wizard.
- Install Vertica on one or more hosts using the command line, then install the Management Console and import the cluster to be managed.

In This Section

Installing Using the Command Line

Although Vertica supports installation on one node, two nodes, and multiple nodes, this section describes how to install the Vertica software on a cluster of nodes. It assumes that you have already performed the tasks in Before You Install Vertica, and that you have a Vertica license key.

To install Vertica, complete the following tasks:

1. Download and install the Vertica server package
2. Installing Vertica with the Installation Script

Special notes

- Downgrade installations are not supported.
Be sure that you download the RPM for the correct operating system and architecture.

Vertica supports two-node clusters with zero fault tolerance (K=0 safety). This means that you can add a node to a single-node cluster, as long as the installation node (the node upon which you build) is not the loopback node (localhost/127.0.0.1).

The Version 7.0 installer introduces new platform verification tests that prevent the install from continuing if the platform requirements are not met by your system. Manually verify that your system meets the requirements in Before You Install Vertica on your systems. These tests ensure that your platform meets the hardware and software requirements for Vertica. Previous versions documented these requirements, but the installer did not verify all of the settings. If this is a fresh install, then you can simply run the installer and view a list of the failures and warnings to determine which configuration changes you must make.

Download and Install the Vertica Server Package

To download and install the Vertica server package:

1. Use a Web browser to log in to myVertica portal.

2. Click the Download tab and download the Vertica server package to the Administration Host.

   Be sure the package you download matches the operating system and the machine architecture on which you intend to install it. In the event of a node failure, you can use any other node to run the Administration Tools later.

3. If you installed a previous version of Vertica on any of the hosts in the cluster, use the Administration Tools to shut down any running database.

   The database must stop normally; you cannot upgrade a database that requires recovery.

4. If you are using sudo, skip to the next step. If you are root, log in to the Administration Host as root (or log in as another user and switch to root).

   $ su - root
   password: root-password
   #
Caution: When installing Vertica using an existing user as the dba, you must exit all UNIX terminal sessions for that user after setup completes and log in again to ensure that group privileges are applied correctly.

After Vertica is installed, you no longer need root privileges. To verify sudo, see **General Hardware and OS Requirements and Recommendations**.

5. Use one of the following commands to run the RPM package installer:

- If you are root and installing an RPM:

  ```
  # rpm -Uvh pathname
  ```

- If you are using sudo and installing an RPM:

  ```
  $ sudo rpm -Uvh pathname
  ```

- If you are using Debian:

  ```
  $ sudo dpkg -i pathname
  ```

  where `pathname` is the Vertica package file you downloaded.

Note: If the package installer reports multiple dependency problems, or you receive the error "**ERROR: You're attempting to install the wrong RPM for this operating system**", then you are trying to install the wrong Vertica server package. Make sure that the machine architecture (32-bit or 64-bit) of the package you downloaded matches the operating system.

**Installing Vertica with the Installation Script**

Run the install script after you have installe the Vertica package. The install script is run on a single node, using a Bash shell, and it copies the Vertica package to all other hosts (identified by the `--hosts` argument) in your planned cluster.

The install script runs several tests on each of the target hosts to verify that the hosts meet the system and performance requirements for a Vertica node. The install script modifies some operating system configuration settings to meet these requirements. Other settings cannot be modified by the install script and must be manually reconfigured.

The installation script takes the following basic parameters:
- A list of hosts on which to install.

- Optionally, the Vertica RPM/DEB path and package file name if you have not pre-installed the server package on other potential hosts in the cluster.

- Optionally, a system user name. If you do not provide a user name, then the install script creates a new system user named dbadmin. If you do provide a username and the username does not exist on the system, then the install script creates that user.

For example:

```bash
# /opt/vertica/sbin/install_vertica --hosts node0001,node0002,node0003 \
   --rpm /tmp/vertica_9.0.x.x86_64.RHEL6.rpm \
   --dba-user mydba
```

Note: The install script sets up passwordless ssh for the administrator user across all the hosts. If passwordless ssh is already set up, the install script verifies that it is functioning correctly.

### Perform a Basic Installation of Vertica

1. As root (or sudo) run the install script. The script must be run by a BASH shell as root or as a user with sudo privileges. You can configure many options when running the install script. See `install_vertica Options` below for the complete list of options.

   If the installer fails due to any requirements not being met, you can correct the issue and then re-run the installer with the same command line options.

   **To perform a basic installation:**

   - **As root:**

     ```bash
     # /opt/vertica/sbin/install_vertica --hosts host_list --rpm package_name --dba-user dba_[username]
     ```

   - **Using sudo:**

     ```bash
     $ sudo /opt/vertica/sbin/install_vertica --hosts host_list --rpm package_name --dba-user dba_[username]
     ```
**Important:** If you place vertica_installation somewhere other than /opt/vertica, you need to create a symlink from that location to /opt/vertica. You need to create this symlink on all nodes in the cluster, otherwise the database will not start.

### Basic Installation Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--hosts host_list</td>
<td>A comma-separated list of IP addresses to include in the cluster; do not include space characters in the list. Examples:</td>
</tr>
<tr>
<td></td>
<td>---hosts 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>---hosts 192.168.233.101,192.168.233.102,192.168.233.103</td>
</tr>
<tr>
<td>Note: Vertica stores only IP addresses in its configuration files. You can provide a hostname to the --hosts parameter, but it is immediately converted to an IP address when the script is run.</td>
<td></td>
</tr>
<tr>
<td>--rpm package_name</td>
<td>The path and name of the Vertica RPM package. Example:</td>
</tr>
<tr>
<td>--deb package_name</td>
<td>---rpm /tmp/vertica_9.0.x.x86_64.RHEL6.rpm</td>
</tr>
<tr>
<td></td>
<td>For Debian and Ubuntu installs, provide the name of the Debian package, for example:</td>
</tr>
<tr>
<td></td>
<td>---deb /tmp/vertica_7.2.x86.deb</td>
</tr>
<tr>
<td>--dba-user dba_username</td>
<td>The name of the Database Administrator system account to create. Only this account can run the Administration Tools. If you omit the --dba-user parameter, then the default database administrator account name is dbadmin.</td>
</tr>
<tr>
<td></td>
<td>This parameter is optional for new installations done as root but must be specified when upgrading or when installing using sudo. If upgrading, use the -u parameter to specify the same DBA account name that you used previously. If installing using sudo, the user must already exist.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Note: If you manually create the user, modify the user's .bashrc file to include the line: PATH=/opt/vertica/bin:$PATH so that the Vertica tools such as vsq1 and admintools can be easily started by the dbadmin user.</td>
<td></td>
</tr>
</tbody>
</table>

2. When prompted for a password to log into the other nodes, provide the requested password. Doing so allows the installation of the package and system configuration on the other cluster nodes.

- If you are root, this is the root password.
- If you are using sudo, this is the sudo user password.

The password does not echo on the command line. For example:

```
Vertica Database 8.1. Installation Tool
Please enter password for root@host01:password
```

3. If the dbadmin user, or the user specified in the argument --dba-user, does not exist, then the install script prompts for the password for the user. Provide the password. For example:

```
Enter password for new UNIX user dbadmin:password
Retype new UNIX password for user dbadmin:password
```

4. Carefully examine any warnings or failures returned by install_ vertica and correct the problems.

For example, insufficient RAM, insufficient network throughput, and too high readahead settings on the filesystem could cause performance problems later on. Additionally, LANG warnings, if not resolved, can cause database startup to fail and issues with VSQL. The system LANG attributes must be UTF-8 compatible. Once you fix the problems, re-run the install script.

5. When installation is successful, disconnect from the Administration Host, as instructed by the script. Then, complete the required post-installation steps.

At this point, root privileges are no longer needed and the database administrator can perform any remaining steps.
Install on a FIPS 140-2 Enabled Machine

Vertica supports the implementation of the Federal Information Processing Standard 140-2 (FIPS). FIPS mode can be enabled in the operating system, specifically Red Hat Enterprise Linux 6.6.

**Note: FIPS enablement on the operating system occurs outside of Vertica.**

During the installation process, the Vertica 8.0 installer detects whether the host is operating in FIPS mode, as follows:

- The installer searches for `/proc/sys/crypto/fips_enabled` and examines its content. If the file exists and contains a '1', the host is operating in FIPS mode and the following message appears:

  `/proc/sys/crypto/fips_enabled exists and contains '1', this is a FIPS system`

- The installer then installs symbolic links to the system `libcrypto.so.10` and `libssl.so.10` in `/opt/vertica/lib`.

- If the file does not exist or does not contain a '1', the installer compares the version numbers of the host `libcrypto.so` and `libssl.so` to the library versions in the Vertica package. If the host library file names have a higher version number, they are linked to the system copies. Otherwise, the host uses the libraries from the Vertica package.

  **Important:** The following message appears if the OpenSSL version is `1.0.1#` (where # indicates versions f to t) or higher like `1.0.2h`

  ```
  No version information available
  ```

  This message indicates that you are using a library different from the one that Vertica was built with. As long as you are using a library that is newer than `1.0.1e` (but not yet 1.1), Vertica operates correctly.

Create Symbolic Links to OpenSSL

During Vertica installation the installer determines what library Vertica needs to execute and sets up the following symbolic links in `/opt/vertica/lib`:

- `libssl.so.10`

- `libcrypto.so.10`
These Vertica-created symbolic links are linked to the actual filenames located in the following locations:

- /lib
- /lib64
- /usr/lib
- /usr/lib64
- /lib/x86_64-linux-gnu
- /lib/i386-linux-gnu
- /usr/lib/x86_64-linux-gnu
- /usr/lib/i386-linux-gnu

During processing, the installer places the actual files in /opt/vertica/lib:

- libssl.so.1.0.1e
- libcrypto.so.1.0.1e

If you build an updated OpenSSL, the default names resulting from the OpenSSL build process are:

- libcrypto.so.1.0.0
- libssl.so.1.0.0

In this case, the installer looks at the installed files and determines that these are a newer version than 1.0.1.e. When you attempt to compile the libraries, compilation fails.

In order for it to be linked automatically, you can do one of the following:

- Rename the libraries to indicate the version from which it was built. The install script can find it.
- Create new symbolic links in opt/vertica/lib that point to the location of the newer OpenSSL libraries.

**Important:** Before creating new symbolic links contact your Customer Experience representative.

Create new symbolic links using the instructions provided by your operating system.
Important: If you create new symbolic links, do NOT run install_vertica again. Doing so overwrites the new symbolic links.

For more information see Implementing FIPS 140-2.

Complete Required Post-Installation Steps

1. Log in to the Database Administrator account on the administration host.

2. Install the License Key

3. Accept the EULA.

4. If you have not already done so, proceed to Getting Started. Otherwise, proceed to Configuring the Database in the Administrator's Guide.

install_vertica Options

The table below details all options available to the install_vertica script. Most options have a long and short form. For example --hosts is interchangeable with -s. The only required options are --hosts/-s and --rpm|--deb|--r.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>Display help for this script.</td>
</tr>
<tr>
<td>--hosts host_list, -s host_list</td>
<td>A comma-separated list of host names or IP addresses to include in the cluster. Do not include spaces in the list. The IP addresses or hostnames must be for unique hosts. You cannot list the same host using multiple IP addresses/hostnames. Examples:</td>
</tr>
<tr>
<td></td>
<td>--hosts host01,host02,host03 -s 192.168.233.101,192.168.233.102,192.168.233.103</td>
</tr>
</tbody>
</table>

Note: If you are upgrading an existing installation of Vertica, be sure to use the same host names that you used previously.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--rpm package_name, --deb package_name, -r package_name</td>
<td>The name of the RPM or Debian package. The install package must be provided if you are installing or upgrading multiple nodes and the nodes do not have the latest server package installed, or if you are adding a new node. The install_vertica and update_vertica scripts serially copy the server package to the other nodes and install the package. If you are installing or upgrading a large number of nodes, then consider manually installing the package on all nodes before running the upgrade script, as the script runs faster if it does not need to serially upload and install the package on each node. Example: <code>--rpm vertica_9.0.x.x86_64.RHEL6.rpm</code></td>
</tr>
</tbody>
</table>
| --data-dir data_directory, -d data_directory | Specify the directory for database data and catalog files. The default is `/home/dbadmin`.  

**Note:** Do not use a shared directory over more than one host for this setting. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory. |
| --temp-dir directory | The temporary directory used for administrative purposes. If it is a directory within `/opt/vertica`, then it will be created by the installer. Otherwise, the directory should already exist on all nodes in the cluster. The location should allow dbadmin write privileges.  
The default is `/tmp`.  

**Note:** This is not a temporary data location for the database. |
<p>| --dba-user dba_username, -u dba_username | The name of the Database Administrator system account to create. Only this account can run the |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| Administration Tools. If you omit the `--dba-user` parameter, then the default database administrator account name is `dbadmin`. This parameter is optional for new installations done as root but must be specified when upgrading or when installing using sudo. If upgrading, use the `-u` parameter to specify the same DBA account name that you used previously. If installing using sudo, the user must already exist. 

**Note:** If you manually create the user, modify the user’s `.bashrc` file to include the line: `PATH=/opt/vertica/bin:$PATH` so that the Vertica tools such as `vsql` and `admintools` can be easily started by the `dbadmin` user. |
| `--dba-group GROUP, -g GROUP` | The UNIX group for DBA users. The default is `verticadba`. |
| `--dba-user-home dba_home_directory, -l dba_home_directory` | The home directory for the database administrator. The default is `/home/dbadmin`. |
| `--dba-user-password dba_password, -p dba_password` | The password for the database administrator account. If not supplied, the script prompts for a password and does not echo the input. |
| `--dba-user-password-disabled` | Disable the password for the `--dba-user`. This argument stops the installer from prompting for a password for the `--dba-user`. You can assign a password later using standard user management tools such as `passwd`. |
| `--spread-logging, -W` | Configures spread to output logging output to `/opt/vertica/log/spread_<hostname>.log`. Does not apply to upgrades. 

**Note:** Do not enable this logging unless directed to by Vertica Analytic Database Technical Support. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ssh-password password, -P password</td>
<td>The password to use by default for each cluster host. If not supplied, and the -i option is not used, then the script prompts for the password if and when necessary and does not echo the input. Do not use with the -i option.</td>
</tr>
<tr>
<td></td>
<td><strong>Special note about password:</strong></td>
</tr>
<tr>
<td></td>
<td>If you run the install_vertica script as root, specify the root password with the -P parameter:</td>
</tr>
<tr>
<td></td>
<td># /opt/vertica/sbin/install_vertica -P &lt;root_passwd&gt;</td>
</tr>
<tr>
<td></td>
<td>If, however, you run the install_vertica script with the sudo command, the password for the -P parameter should be the password of the user who runs install_vertica, not the root password. For example if user dbadmin runs install_vertica with sudo and has a password with the value dbapassword, then the value for -P should be dbapassword:</td>
</tr>
<tr>
<td></td>
<td>$ sudo /opt/vertica/sbin/install_vertica -P dbapassword</td>
</tr>
<tr>
<td>--ssh-identity file, -i file</td>
<td>The root private-key file to use if passwordless ssh has already been configured between the hosts. Verify that normal SSH works without a password before using this option. The file can be private key file (for example, id_rsa), or PEM file. Do not use with the --ssh-password/ -P option. Vertica accepts the following:</td>
</tr>
<tr>
<td></td>
<td>- By providing an SSH private key which is not password protected. You cannot run the install_vertica script with the sudo command when using this method.</td>
</tr>
<tr>
<td></td>
<td>- By providing a password-protected private key and using an SSH-Agent. Note that sudo typically resets environment variables when it is invoked. Specifically, the SSH_AUTHSOCK variable required by the SSH-Agent may be reset. Therefore, configure your system to</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>maintain SSH_AUTHSOCK or invoke the install_vertica command using a method similar to the following: sudo SSH_AUTHSOCK=$SSH_AUTHSOCK /opt/vertica/sbin/install_vertica ...</td>
<td></td>
</tr>
<tr>
<td>--config-file file, -z file</td>
<td>Accepts an existing properties file created by --record-config file_name. This properties file contains key/value parameters that map to values in the install_vertica script, many with Boolean arguments that default to false.</td>
</tr>
</tbody>
</table>
| --add-hosts host_list, -A host_list | A comma-separated list of hosts to add to an existing Vertica cluster. --add-hosts modifies an existing installation of Vertica by adding a host to the database cluster and then reconfiguring the spread. This is useful for increasing system performance or setting K-safety to one (1) or two (2). Notes:  
  - If you have used the -T parameter to configure spread to use direct point-to-point communication within the existing cluster, you must use the -T parameter when you add a new host; otherwise, the new host automatically uses UDP broadcast traffic, resulting in cluster communication problems that prevent Vertica from running properly.  
    Examples:  
    ```bash  
    --add-hosts host01  
    --add-hosts 192.168.233.101  
    ```  
  - The update_vertica script described in Adding Nodes calls the install_vertica script to update the installation. You can use either the install_vertica or update_
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertica</td>
<td>vertica script with the --add-hosts parameter.</td>
</tr>
<tr>
<td>--record-config file_name, -B file_name</td>
<td>Accepts a file name, which when used in conjunction with command line options, creates a properties file that can be used with the --config-file parameter. This parameter creates the properties file and exits; it has no impact on installation.</td>
</tr>
<tr>
<td>--clean</td>
<td>Forcibly cleans previously stored configuration files. Use this parameter if you need to change the hosts that are included in your cluster. Only use this parameter when no database is defined. Cannot be used with update_vertica.</td>
</tr>
<tr>
<td>--license { license_file</td>
<td>CE }, -L { license_file</td>
</tr>
<tr>
<td></td>
<td>--license CE</td>
</tr>
<tr>
<td></td>
<td>--license /tmp/vlicense.dat</td>
</tr>
<tr>
<td>--remove-hosts host_list, -R host_list</td>
<td>A comma-separated list of hosts to remove from an existing Vertica cluster. --remove-hosts modifies an existing installation of Vertica by removing a host from</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>the database cluster and then reconfiguring the spread. This is useful for removing an obsolete or over-provisioned system. For example:</td>
</tr>
<tr>
<td>---remove-hosts host01</td>
<td></td>
</tr>
<tr>
<td>-R 192.168.233.101</td>
<td></td>
</tr>
<tr>
<td>Notes:</td>
<td></td>
</tr>
<tr>
<td>• If you used the -T parameter to configure spread to use direct point-to-point communication within the existing cluster, you must use -T when you remove a host; otherwise, the hosts automatically use UDP broadcast traffic, resulting in cluster communication problems that prevents Vertica from running properly.</td>
<td></td>
</tr>
<tr>
<td>• The update_vertica script described in Removing Nodes in the Administrator's Guide calls the install_vertica script to perform the update to the installation. You can use either the install_vertica or update_vertica script with the -R parameter.</td>
<td></td>
</tr>
<tr>
<td>--control-network { BCAST_ADDR</td>
<td>default }, -S { BCAST_ADDR</td>
</tr>
<tr>
<td>Note: The --control-network must match the subnet for at least some of the nodes in the database. If the provided address does not match the subnet of any node in the database then the installer displays an error and stops. If the provided address matches some, but not all of the node's subnets, then</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>a warning is displayed, but the install continues. Ideally, the value for --control-network should match all node subnets.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>--control-network default</td>
</tr>
<tr>
<td></td>
<td>--control-network 10.20.100.255</td>
</tr>
</tbody>
</table>

| --point-to-point, -T | Configures spread to use direct point-to-point communication between all Vertica nodes. You should use this option if your nodes aren't located on the same subnet. You should also use this option for all virtual environment installations, regardless of whether the virtual servers are on the same subnet or not. The maximum number of spread daemons supported in point-to-point communication in Vertica is 80. It is possible to have more than 80 nodes by using large cluster mode, which does not install a spread daemon on each node. |
|                     | Cannot be used with --broadcast, as the setting must be either --broadcast or --point-to-point. |
|                     | **Important:** When changing the configuration from --broadcast (the default) to --point-to-point or from --point-to-point to --broadcast, the --control-network parameter must also be used. |
|                     | **Note:** Spread always runs on UDP. -T does not denote TCP. |

<p>| --broadcast, -U | Specifies that Vertica use UDP broadcast traffic by spread between nodes on the subnet. This parameter is automatically used by default. No more than 80 spread daemons are supported by broadcast traffic. It is possible to have more than 80 nodes by using large cluster mode, which does |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not install a spread daemon on each node. Cannot be used with --point-to-point, as the setting must be either --broadcast or --point-to-point. Important: When changing the configuration from --broadcast (the default) to --point-to-point or from --point-to-point to --broadcast, the --control-network parameter must also be used. Note: Spread always runs on UDP. -U does not mean use UDP instead of TCP.</td>
</tr>
<tr>
<td>--accept-eula, -Y</td>
<td>Silently accepts the EULA agreement. On multi-node installations, the --accept-eula value is propagated throughout the cluster at the end of the installation, at the same time as the Administration Tools metadata. Use the --accept-eula option with the --license option to activate your license.</td>
</tr>
<tr>
<td>--no-system-configuration</td>
<td>By default, the installer makes system configuration changes to meet server requirements. If you do not want the installer to change any system properties, then use the --no-system-configuration. The installer presents warnings or failures for configuration settings that do not meet requirements that it normally would have automatically configured. Note: The system user account is still created/updated when using this parameter.</td>
</tr>
</tbody>
</table>
| --failure-threshold       | Stops the installation when the specified failure threshold is encountered. Options can be one of:  
  * HINT - Stop the install if a HINT or greater issue
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>is encountered during the installation tests. HINT configurations are settings you should make, but the database runs with no significant negative consequences if you omit the setting.</td>
</tr>
<tr>
<td></td>
<td>• WARN (default) - Stop the installation if a WARN or greater issue is encountered. WARN issues may affect the performance of the database. However, for basic testing purposes or Community Edition users, WARN issues can be ignored if extreme performance is not required.</td>
</tr>
<tr>
<td></td>
<td>• FAIL - Stop the installation if a FAIL or greater issue is encountered. FAIL issues can have severely negative performance consequences and possible later processing issues if not addressed. However, Vertica can start even if FAIL issues are ignored.</td>
</tr>
<tr>
<td></td>
<td>• HALT - Stop the installation if a HALT or greater issue is encountered. The database may not be able to be started if you choose his option. Not supported in production environments.</td>
</tr>
<tr>
<td></td>
<td>• NONE - Do not stop the installation. The database may not start. Not supported in production environments.</td>
</tr>
<tr>
<td>--large-cluster, -2</td>
<td>Enables a large cluster layout, in which control message responsibilities are delegated to a subset of Vertica Analytic Database nodes (called control nodes) to improve control message performance in large clusters. Consider using this parameter with more than 50 nodes. Options can be one of:</td>
</tr>
<tr>
<td>[ &lt;integer&gt;</td>
<td>DEFAULT ]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>120</td>
<td>120 for all new databases.</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT—Vertica Analytic Database chooses the number of control nodes using calculations based on the total number of cluster nodes in the -hosts argument.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Large Cluster in the Administrator's Guide.</td>
</tr>
</tbody>
</table>

## Installing Vertica Silently

This section describes how to create a properties file that lets you install and deploy Vertica-based applications quickly and without much manual intervention.

**Note:** The procedure assumes that you have already performed the tasks in Before You Install Vertica.

### Install the properties file:

1. Download and install the Vertica install package, as described in Installing Vertica.

2. Create the properties file that enables non-interactive setup by supplying the parameters you want Vertica to use. For example:

   The following command assumes a multi-node setup:

   ```
   # /opt/vertica/sbin/install_vertica --record-config file_name --license /tmp/license.txt --accept-eula \
   # --dba-user-password password --ssh-password password --hosts host_list --rpm package_name
   ```

   The following command assumes a single-node setup:

   ```
   # /opt/vertica/sbin/install_vertica --record-config file_name --license /tmp/license.txt --accept-eula \
   # --dba-user-password password
   ```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--record-file file_name</td>
<td>[Required] Accepts a file name, which when used in conjunction with command line options, creates a properties file that can be used with the --config-file option during setup. This flag creates the properties file and exits; it has no impact on installation.</td>
</tr>
<tr>
<td>--license { license_file</td>
<td>CE</td>
</tr>
<tr>
<td>--accept-eula</td>
<td>Silently accepts the EULA agreement during setup.</td>
</tr>
<tr>
<td>--dba-user-password password</td>
<td>The password for the Database Administrator account; if not supplied, the script prompts for the password and does not echo the input.</td>
</tr>
<tr>
<td>--ssh-password password</td>
<td>The root password to use by default for each cluster host; if not supplied, the script prompts for the password if and when necessary and does not echo the input.</td>
</tr>
<tr>
<td>--hosts host_list</td>
<td>A comma-separated list of hostnames or IP addresses to include in the cluster; do not include space characters in the list. Examples: --hosts host01,host02,host03 --hosts 192.168.233.101,192.168.233.102,192.168.233.103</td>
</tr>
<tr>
<td>--rpm package_name --deb package_name</td>
<td>The name of the RPM or Debian package that contained this script. Example:</td>
</tr>
</tbody>
</table>
### Option | Description
--- | ---
--rpm vertica_9.0.x.x86_64.RHEL6.rpm
This parameter is required on multi-node installations if the RPM or DEB package is not already installed on the other hosts.

See [Installing Vertica with the Installation Script](#) for the complete set of installation parameters.

**Tip:** Supply the parameters to the properties file once only. You can then install Vertica using just the --config-file parameter, as described below.

3. Use one of the following commands to run the installation script.

- If you are root:

  ```bash
  /opt/vertica/sbin/install_vertica --config-file file_name
  ```

- If you are using sudo:

  ```bash
  $ sudo /opt/vertica/sbin/install_vertica --config-file file_name
  ```

--config-file file_name accepts an existing properties file created by --record-config file_name. This properties file contains key/value parameters that map to values in the install_vertica script, many with boolean arguments that default to false.

The command for a single-node install might look like this:

```bash
# /opt/vertica/sbin/install_vertica --config-file /tmp/vertica-inst.prp
```

4. If you did not supply a --ssh-password password parameter to the properties file, you are prompted to provide the requested password to allow installation of the RPM/DEB and system configuration of the other cluster nodes. If you are root, this is the root password. If you are using sudo, this is the sudo user password. The password does not echo on the command line.

**Note:** If you are root on a single-node installation, you are not prompted for a password.
5. If you did not supply a `--dba-user-password` password parameter to the properties file, you are prompted to provide the database administrator account password.

The installation script creates a new Linux user account (dbadmin by default) with the password that you provide.

6. Carefully examine any warnings produced by `install_vertica` and correct the problems if possible. For example, insufficient RAM, insufficient Network throughput and too high readahead settings on filesystem could cause performance problems later on.

   **Note:** You can redirect any warning outputs to a separate file, instead of having them display on the system. Use your platforms standard redirected mechanisms. For example: `install_vertica [options] > /tmp/file 1>&2`.

7. Optionally perform the following steps:
   - Install the ODBC and JDBC driver.
   - Install the vsql client application on non-cluster hosts.

8. Disconnect from the Administration Host as instructed by the script. This is required to:
   - Set certain system parameters correctly.
   - Function as the Vertica database administrator.

At this point, Linux root privileges are no longer needed. The database administrator can perform the remaining steps.

   **Note:** When creating a new database, the database administrator might want to use different data or catalog locations than those created by the installation script. In that case, a Linux administrator might need to create those directories and change their ownership to the database administrator.

- If you supplied the `--license` and `--accept-eula` parameters to the properties file, then proceed to the **Getting Started** and then see **Configuring the Database** in the Administrator's Guide. Otherwise:

1. Log in to the Database Administrator account on the administration host.

2. Accept the End User License Agreement and install the license key you downloaded previously as described in **Install the License Key**.
3. Proceed to Getting Started and then see Configuring the Database in the Administrator's Guide.

Notes

- Downgrade installations are not supported.
- The following is an example of the contents of the configuration properties file:

```
accept_eula = True
license_file = /tmp/license.txt
record_to = file_name
root_password = password
vertica_dba_group = verticadba
vertica_dba_user = dbadmin
vertica_dba_user_password = password
```

Installing Vertica on Amazon Web Services (AWS)

Beginning with Vertica 6.1.x, you can use Vertica on AWS by utilizing a pre-configured Amazon Machine Image (AMI). For details on installing and configuring a cluster on AWS, refer to Installing and Running Vertica on AWS.

Installing and Configuring Management Console

This section describes how to install, configure, and upgrade Management Console (MC). If you need to back up your instance of MC, see Backing Up MC in the Administrator's Guide.

You can install MC before or after you install Vertica; however, consider installing Vertica and creating a database before you install MC.

In This Section
Before You Install MC

Management Console (MC) 9.0.x is compatible with the latest hotfix version of Vertica server 7.2.3 and above. Read the following documents for more information:

- Supported Platforms document, at http://my.vertica.com/docs. The Supported Platforms document also lists supported browsers for MC.

- Installation Overview and Checklist. Make sure you have everything ready for your Vertica configuration.

- Before You Install Vertica. Read for required prerequisites for all Vertica configurations, including Management Console.

Driver Requirements for Linux SuSe Distributions

The MC (vertica-console) package contains the Oracle Implementation of Java 7 JRE and requires that you install the unixODBC driver manager on SuSe Linux platforms. unixODBC provides needed libraries libodbc and lidodbcinst.

Port Requirements

When you use MC to create a Vertica cluster, the Create Cluster Wizard uses SSH on its default port (22).

Port 5444 is the default agent port and must be available for MC-to-node and node-to-node communications.

Port 5450 is the default MC port and must be available for node-to-MC communications.

See Ensure Ports Are Available for more information about port and firewall considerations.

Firewall Considerations

Make sure that a firewall or iptables are not blocking communications between the cluster's database, Management Console, and MC's agents on each cluster node.
IP Address Requirements

If you install MC on a server outside the Vertica cluster it will be monitoring, that server must be accessible to at least the public network interfaces on the cluster.

Hardware Requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>CPU</th>
<th>RAM</th>
<th>Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>4-core</td>
<td>4G</td>
<td>2G</td>
</tr>
<tr>
<td>Recommended</td>
<td>8-core</td>
<td>8G</td>
<td>2G</td>
</tr>
</tbody>
</table>

You can install MC on any node in the cluster, or its own dedicated node. When running the MC on a node in the cluster, note that MC shares RAM and time on CPU cores with other Vertica processes. See Disk Space Requirements for Vertica.

Time Synchronization and MC's Self-Signed Certificate

When you connect to MC through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See Set Up Time Synchronization in Installing Vertica.

SSL Requirements

The openssl package must be installed on your Linux environment so SSL can be set up during the MC configuration process. See SSL Overview in the Administrator's Guide.

File Permission Requirements

On your local workstation, you must have at least read/write privileges on any files you plan to upload to MC through the Cluster Installation Wizard. These files include the Vertica server...
package, the license key (if needed), the private key file, and an optional CSV file of IP addresses.

Monitor Resolution

Management Console requires a minimum resolution of 1024 x 768, but Vertica recommends higher resolutions for optimal viewing.

Installing Management Console

You can install Management Console on any node you plan to include in the Vertica database cluster, as well as on its own, dedicated server outside the cluster.

Install Management Console on the MC Server

1. Download the MC package from the myVertica portal:
   - vertica-console-current-version.Linux-distro
   - Save the package to a location on the target server, such as /tmp.

2. On the target server, log in as root or a user with sudo privileges.

3. Change to the directory where you saved the MC package.

4. Install MC using your local Linux distribution package management system—rpm, yum, zypper, apt, dpkg. For example:

   Red Hat 6
   - # rpm -Uvh vertica-console-current-version.x86_64.RHEL6.rpm

   Debian and Ubuntu
   - # dpkg -i vertica-console-current-version.deb

5. If you stopped the database before upgrading MC, restart the database.

   As the root user, use the following command:
For versions of Red Hat 7/CentOS 7 and above, run:

```bash
# systemctl start vertica-consoled
```

6. Open a browser and enter the URL of the MC installation, one of the following:

- **IP address:**
  ```
  https://ip-address:mc-port/
  ```

- **Server host name:**
  ```
  https://hostname:mc-port/
  ```

By default, `mc-port` is 5450.

7. If MC was not previously configured, the Configuration Wizard dialog box appears. Configuration steps are described in Configuring MC.

   If MC was previously configured, Vertica prompts you to accept the end-user license agreement (EULA) when you first log in to MC after the upgrade.

### Configuring MC

After you install MC, you need to configure it through a client browser connection. An MC configuration wizard walks you through creating the Linux MC super administrator account, storage locations, and other settings that MC needs to run. Information you provide during the configuration process is stored in the `/opt/vconsole/config/console.properties` file.

If you need to change settings after the configuration wizard ends, such as port assignments, you can do so later through Home > MC Settings page.
How to Configure MC

1. Open a browser session.

2. Enter the IP address or host name of the server on which you installed MC (or any cluster node's IP/host name if you already installed Vertica), and include the default MC port 5450. For example, you'll enter one of:


3. Follow the configuration wizard.

About Authentication for the MC Super Administrator

In the final step of the configuration process, you choose an authentication method for the MC super administrator. You can decide to have MC authenticate the MC super (in which case the process is complete), or you can choose LDAP.

If you choose LDAP, provide the following information for the newly-created MC super administrator:

- Corporate LDAP service host (IP address or host name)
- LDAP server running port (default 389)
- LDAP DN (distinguished name) for base search/lookup/authentication criteria
  
  At a minimum, specify the dc (domain component) field. For example: dc=vertica, dc=com generates a unique identifier of the organization, like the corporate Web URL vertica.com

- Default search path for the organization unit (ou)
  
  For example: ou=sales, ou=engineering

- Search attribute for the user name (uid), common name (cn), and so on
  
  For example, uid=jdoe, cn=Jane Doe

- Binding DN and password for the MC super administrator.
In most cases, you provide the "Bind as administrator" fields, information used to establish the LDAP service connection for all LDAP operations, like search. Instead of using the administrator user name and password, the MC administrator could use his or her own LDAP credentials, as long as that user has search privileges.

If You Choose Bind Anonymously

Unless you specifically configure the LDAP server to deny anonymous binds, the underlying LDAP protocol will not cause MC's Configure Authentication process to fail if you choose "Bind anonymously" for the MC administrator. Before you use anonymous bindings for LDAP authentication on MC, be sure that your LDAP server is configured to explicitly disable/enable this option. For more information, see the article on Infusion Technology Solutions and the OpenLDAP documentation on access control.

What Happens Next

Shortly after you click Finish, you should see a status in the browser; however, for several seconds you might see only an empty page. During this brief period, MC runs as the local user 'root' long enough to bind to port number 5450. Then MC switches to the MC super administrator account that you just created, restarts MC, and displays the MC login page.

Where to Go Next

If you are a new MC user and this is your first MC installation, you might want to familiarize yourself with MC design. See Management Console in Vertica Concepts.

If you'd rather use MC now, the following following topics in the Administrator's Guide should help get you started:

<table>
<thead>
<tr>
<th>If you want to ...</th>
<th>See ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the MC interface to install Vertica on a cluster of hosts</td>
<td>Creating a Cluster Using MC</td>
</tr>
<tr>
<td>Create a new, empty Vertica database or import an existing Vertica database cluster into the MC interface</td>
<td>Managing Database Clusters</td>
</tr>
<tr>
<td>Create new MC users and map them to one or more Vertica databases that you manage through the MC</td>
<td>Managing Users and Privileges (About MC Users and About MC Privileges)</td>
</tr>
</tbody>
</table>
If you want to ... | See ...
---|---
interface | and Roles)
Monitor MC and one or more MC-managed Vertica databases | Monitoring Vertica Using Management Console
Change default port assignments or upload a new Vertica license or SSL certificate | Managing MC Settings
Compare MC functionality to functionality that the Administration Tools provides | Administration Tools and Management Console

## Installing Vertica in Eon Mode Beta

Eon Mode Beta separates the computational processes of Vertica from its storage layer, allowing you to elastically scale your computational resources as your workload changes. You can only Eon Mode Beta on Amazon Web Services.

**Important:** This functionality is currently in Beta. Vertica does not support running Eon Mode Beta in a production environment.

Running Vertica in Eon Mode Beta might be a good choice for you in the following situations:

- You are deploying Vertica in the AWS cloud.
- You have variable workloads.
- You need to elastically scale your database resources.

You can install Vertica with Eon Mode Beta using in-browser wizards provided by Vertica Management Console with Provisioning. If you have more specific AWS configuration needs, you can instead use the Administration Tools or the command line to install Vertica on AWS resources and create an Eon Mode Beta database.

## Using Vertica Management Console with Provisioning

Through the AWS Marketplace, you can launch Vertica Management Console with Provisioning. Vertica Management Console with Provisioning provides in-browser wizards you can follow to deploy Vertica cluster instances and create an Eon Mode Beta database on them.

See [Installing Using Management Console with Provisioning](#).

Vertica Analytic Database (9.0.x)
Using the Command Line and the Administration Tools

Choose this method of installation if you are familiar with configuring AWS and have many specific AWS configuration needs.

This method consists of downloading the Vertica server package and installing Vertica using the command line. The installation process for Eon Mode Beta is similar to installing Vertica in Enterprise Mode on AWS resources.

After installing Vertica, create an Eon Mode Beta database using the Administration Tools. See Installing Vertica in Eon Mode Beta Using the Command Line.

Installing Using Management Console with Provisioning

Management Console with Provisioning is a deployment of Vertica Management Console (MC) that allows you to provision, manage, and monitor Vertica on AWS resources.

You can launch Vertica Management Console with Provisioning and its associated AWS resources through the AWS Marketplace. Use MC to deploy the Vertica cluster instances and create an Eon Mode Beta or Enterprise Mode database.

Prerequisites

- AWS account with permissions to create a VPC, subnet, security group, EC2 instances, and IAM roles
- Amazon key pair for SSH access to an EC2 instance

Deploy the Management Console with Provisioning

Starting the in the AWS Marketplace, you will launch the provisioning instance from which you can install Vertica.

1. Log in to the AWS Marketplace with an AWS account (see the Prerequisites section above).

2. Search for "Vertica" in the AWS Marketplace and select one of the following options. You can launch Vertica Management Console with Provisioning resources on either a Red Hat
or CentOS operation system:

- Vertica Management Console with Provisioning, Red Hat
- Vertica Management Console with Provisioning, CentOS

3. On the Launch page, choose to create a VPC or install on an existing one:
   - Basic. This option creates a VPC in which to install the Vertica Provisioning Service and related AWS resources.
   - Advanced. Use this option to install the Vertica Provisioning Service and AWS resources in an existing VPC.

4. Click Launch with CloudFormation Console. The CloudFormation Console opens.

5. The Cloudformation Console automatically supplies the URL in the Specify an Amazon S3 template URL field. Click Next.

6. Follow the CloudFormation workflow and enter the parameters for your stack configuration.

   Important: Take note of the username and password you set for Management Console during this step. You cannot recover or reset these credentials after you create the stack.

7. After confirming the details you have provided for your new stack, click Create. The AWS console brings you to the Stacks page, where you can view the progress of the creation process. The process takes several minutes.

8. The Outputs tab displays information about accessing your environment after the process completes.

**Access Management Console with Provisioning**

1. On the CloudFormation Stacks page, select your new stack and view the Outputs tab. This tab provides information about accessing your environment, as well as documentation and licensing resources.
2. Click the Access Management Console URL. This link takes you to the MC login page.

3. To log in, enter the MC username and password that you created using the CloudFormation Console.

   After login, MC displays the home page, with options to provision a new cluster or database or import existing ones. This page also provides a Resources section with links to Vertica online training, blogs, community, and help resources.
Provision a New Vertica Cluster and Database

You will provision new Vertica clusters in the same VPC and subnet as the Vertica Management Console instance.

**Note:** If you choose EBS volumes for your cluster, make sure your AWS account's EC2 service limits for IOPS and volume storage are high enough to use those volumes. Check these limits on the AWS console EC2 dashboard. If the limits on your account are not sufficient for the volumes you choose, the provisioning process rolls back.

1. On the MC home page, click Create a New Vertica Database Cluster. The creation wizard appears in a new dialog.

2. Choose a mode for your database: Eon Mode Beta or Enterprise Mode. Note that you cannot switch the mode on your database after creation.

3. Follow the cluster creation wizard and enter the parameters for your new cluster and database. You will use the authentication method you selected in the Cloudformation Console when launching MC with Provisioning.

   If you choose Enterprise Mode, the wizard offers two paths:
   - Quick Create provides default values during cluster creation.
   - Custom Create allows you to specify EC2 instance types and other AWS resources for your Vertica cluster instances.

4. After confirming the details you have provided for your new cluster, click Create. The dialog shows you the progress of the creation process, which takes a few minutes. (You can leave or close the browser during this process; to return to this progress window, select Create a New Vertica Database Cluster on the home page.)

   **Note:** During Eon Mode Beta database creation, use an existing Amazon S3 bucket in the same region as your instances for your communal storage location. Specify a new subfolder name, which Vertica will dynamically create within the existing S3 bucket. For example, s3://existingbucket/newstorage1. You can create a new subfolder within existing ones, but database creation will roll back if you do not specify any new subfolder name.

5. The dialog displays a success message when the creation process completes. Click Get Started to view the Fast Tasks page.
View your new database any time under the Available Databases section of the Management Console home page. See Managing Database Clusters for more about further managing your cluster, instances, and database using MC.

See Also

- Using Eon Mode Beta
- Using Management Console
- Managing Database Clusters

Installing Vertica in Eon Mode Beta Using the Command Line

The command line installation process for Eon Mode Beta is similar to installing Vertica in Enterprise Mode on AWS resources. In the steps below, prepare a cluster on AWS resources and install Vertica on it using the referenced guides.

Then create an Eon Mode Beta database using Administration Tools command line options.

Install a Vertica Cluster on AWS Resources

For more information about preparing Vertica clusters in AWS, see the full Vertica on Amazon Web Services guide.

1. Prepare a cluster on AWS using the steps detailed in Installing and Running Vertica on AWS. Make sure that all of the instances are in the same IAM security group.

2. Review and configure your nodes so they meet the requirements defined in the Installation Overview and Checklist.

3. Follow the instructions in Installing Using the Command Line.

4. Perform any additional tasks in After You Install Vertica.
Create an Eon Mode Beta Database Using the Administration Tools

This is the stage where creating an Eon Mode Beta database differs from creating a database in Enterprise Mode.

For a short overview of using the Administration Tools for the first time after installing Vertica, refer to Creating a Database in Installing Vertica guide. This includes how to access the Administration Tools and install your license.

For a detailed walkthrough of the database creation wizard, follow the steps in the Create an Eon Mode Beta Database section of Creating a Database in the Administrator's Guide.

See Also

- Using Eon Mode Beta
Creating a Cluster Using MC

You can use Management Console to install a Vertica cluster on hosts where Vertica software has not been installed. The Cluster Installation wizard lets you specify the hosts you want to include in your Vertica cluster, loads the Vertica software onto the hosts, validates the hosts, and assembles the nodes into a cluster.

Management Console must be installed and configured before you can create a cluster on targeted hosts. See Installing and Configuring the MC for details.

Steps Required to Install a Vertica Cluster Using MC:

- Install and configure MC
- Prepare the Hosts
- Create the private key file and copy it to your local machine
- Run the Cluster Installation Wizard
- Validate the hosts and create the cluster
- Create a new database on the cluster

Prepare the Hosts

Before you can install a Vertica cluster using the MC, you must prepare each host that will become a node in the cluster. The cluster creation process runs validation tests against each host before it attempts to install the Vertica software. These tests ensure that the host is correctly configured to run Vertica.

Install Perl

The MC cluster installer uses Perl to perform the installation. Install Perl 5 on the target hosts before performing the cluster installation. Perl is available for download from www.perl.org.
Validate the Hosts

The validation tests provide:

- Warnings and error messages when they detect a configuration setting that conflicts with the Vertica requirements or any performance issue
- Suggestions for configuration changes when they detect an issue

**Note:** The validation tests do not automatically fix all problems they encounter.

All hosts must pass validation before the cluster can be created.

If you accepted the default configuration options when installing the OS on your host, then the validation tests will likely return errors, since some of the default options used on Linux systems conflict with Vertica requirements. See Installing Vertica for details on OS settings. To speed up the validation process you can perform the following steps on the prospective hosts before you attempt to validate the hosts. These steps are based on Red Hat Enterprise Linux and CentOS systems, but other supported platforms have similar settings.

On each host you want to include in the Vertica cluster, you must stage the host according to [Before You Install Vertica](#).

Create a Private Key File

Before you can install a cluster, Management Console must be able to access the hosts on which you plan to install Vertica. MC uses password-less SSH to connect to the hosts and install Vertica software using a private key file.

If you already have a private key file that allows access to all hosts in the potential cluster, you can use it in the cluster creation wizard.

**Note:** The private key file is required to complete the MC cluster installation wizard.

Create a Private Key File

1. Log into the server as root or as a user with sudo privileges.
2. Change to your home directory.
3. Create an .ssh directory if one does not already exist.

   $ mkdir .ssh

4. Generate a passwordless private key/public key pair.

   $ ssh-keygen -q -t rsa -f ~/.ssh/vid_rsa -N ''

   This command creates two files: vid_rsa and vid_rsa.pub. The vid_rsa file is the private key file that you upload to the MC so that it can access nodes on the cluster and install Vertica. The vid_rsa.pub file is copied to all other hosts so that they can be accessed by clients using the vid_rsa file.

5. Make your .ssh directory readable and writable only by yourself.

   $ chmod 700 /root/.ssh

6. Change to the .ssh directory.

   $ cd ~/.ssh

7. Edit sshd.config as follows to disable password authentication for root:

   PermitRootLogin without-password

8. Concatenate the public key into to the file vauthorized_keys2.

   $ cat vid_rsa.pub >> vauthorized_keys2

9. If the host from which you are creating the public key will also be in the cluster, copy the public key into the local-hosts authorized key file:

   cat vid_rsa.pub >> authorized_keys

10. Make the files in your .ssh directory readable and writable only by yourself.

    $ chmod 600 ~/.ssh/*
11. Create the .ssh directory on the other nodes.

   $ ssh <host> "mkdir /root/.ssh"

12. Copy the vauthorized key file to the other nodes.

   $ scp -r /root/.ssh/vauthorized_keys2 <host>:/root/.ssh/

13. On each node, concatenate the vauthorized_keys2 public key to the authorized_keys file and make the file readable and writable only by the owner.

   $ ssh <host> "cd /root/.ssh/;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"

14. On each node, remove the vauthorized_keys2 file.

   $ ssh -i /root/.ssh/vid_rsa <host> "rm /root/.ssh/vauthorized_keys2"

15. Copy the vid_rsa file to the workstation from which you will access the MC cluster installation wizard. This file is required to install a cluster from the MC.

   A complete example of the commands for creating the public key and allowing access to three hosts from the key is below. The commands are being initiated from the docg01 host, and all hosts will be included in the cluster (docg01 - docg03):

   ssh docg01
cd ~/.ssh
ssh-keygen -q -t rsa -f ~/.ssh/vid_rsa -N ''
cat vid_rsa.pub > vauthorized_keys2
cat vid_rsa.pub >> authorized_keys
chmod 600 ~/.ssh/*
scp -r /root/.ssh/vauthorized_keys2 docg02:/root/.ssh/.
scp -r /root/.ssh/vauthorized_keys2 docg03:/root/.ssh/.
ssh docg02 "cd /root/.ssh/;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"
ssh docg03 "cd /root/.ssh/;cat vauthorized_keys2 >> authorized_keys; chmod 600 /root/.ssh/authorized_keys"
ssh -i /root/.ssh/vid_rsa docg02 "rm /root/.ssh/vauthorized_keys2"
ssh -i /root/.ssh/vid_rsa docg03 "rm /root/.ssh/vauthorized_keys2"
rm ~/.ssh/vauthorized_keys2

**Use the MC Cluster Installation Wizard**

The Cluster Installation Wizard guides you through the steps required to install a Vertica cluster on hosts that do not already have Vertica software installed.
Prerequisites

Before you proceed, make sure you:

- **Installed and configured MC.**

- **Prepared the hosts** that you will include in the Vertica database cluster.

- **Created the private key (pem) file** and copied it to your local machine.

- Obtained a copy of your Vertica license if you are installing the Premium Edition. If you are using the Community Edition, a license key is not required.

- Downloaded the Vertica server RPM (or DEB file).

- Have read/copy permissions on files stored on the local browser host that you will transfer to the host on which MC is installed.

Permissions on Files to Transfer to MC

On your local workstation, you must have at least read/write privileges on files you'll upload to MC through the Cluster Installation Wizard. These files include the Vertica server package, the license key (if needed), the private key file, and an optional CSV file of IP addresses.

Create a New Vertica Cluster Using MC

1. Connect to Management Console and log in as an MC administrator.

2. On MC's Home page, click the Provisioning task. The Provisioning dialog appears.

3. Click Create a new cluster.

4. The Create Cluster wizard opens. Provide the following information:
a. Cluster name—A label for the cluster

b. Vertica Admin User—The user that is created on each of the nodes when they are
installed, typically 'dbadmin'. This user has access to Vertica and is also an OS user on
the host.

c. Password for the Vertica Admin User—The password you enter (required) is set for
each node when MC installs Vertica.

   Note: MC does not support an empty password for the administrative user.

d. Vertica Admin Path—Storage location for catalog files, which defaults to
/home/dbadmin unless you specified a different path during MC configuration (or later
on MC's Settings page).

   Important: The Vertica Admin Path must be the same as the Linux database
   administrator's home directory. If you specify a path that is not the Linux
dbadmin's home directory, MC returns an error.

5. Click Next and specify the private key file and host information:

a. Click Browse and navigate to the private key file (vid_rsa) that you created earlier.

   Note: You can change the private key file at the beginning of the validation stage
   by clicking the name of the private key file in the bottom-left corner of the page.
   However, you cannot change the private key file after validation has begun unless
   the first host fails validation due to an SSH login error.

b. Include the host IP addresses. You have three options:

   Specify later (but include number of nodes). This option allows you to specify the
   number of nodes, but not the specific IPs. You can specify the specific IPs before you
   validate hosts.

   Import IP addresses from local file. You can specify the hosts in a CSV file using either IP
   addresses or host names.

   Enter a range of IP addresses. You can specify a range of IPs to use for new nodes. For
   example 192.168.1.10 to 192.168.1.30. The range of IPs must be on the same or
   contiguous subnets.
6. Click Next and select the software and license:
   
a. Vertica Software. If one or more Vertica packages have been uploaded, you can select one from the list. Otherwise, select Upload a new local vertica binary file and browse to a Vertica server file on your local system.

b. Vertica License. Click Browse and navigate to a local copy of your Vertica license if you are installing the Premium Edition. Community Edition versions require no license key.

7. Click Next. The Create cluster page opens. If you did not specify the IP addresses, select each host icon and provide an IP address by entering the IP in the box and clicking Apply for each host you add.

You are now ready to Validate Hosts and Create the Cluster.

Validate Hosts and Create the Cluster

Host validation is the process where the MC runs tests against each host in a proposed cluster. You can validate hosts only after you have completed the cluster installation wizard. You must validate hosts before the MC can install Vertica on each host.

At any time during the validation process, but before you create the cluster, you can add and remove hosts by clicking the appropriate button in the upper left corner of the page on MC. A Create Cluster button appears when all hosts that appear in the node list are validated.

How to Validate Hosts

To validate one or more hosts:

1. Connect to Management Console and log in as an MC administrator.

2. On the MC Home page, click the Databases and Clusters task.

3. In the list of databases and clusters, select the cluster on which you have recently run the cluster installation wizard (Creating... appears under the cluster) and click View.

4. Validate one or several hosts:
   
   - To validate a single host, click the host icon, then click Validate Host.
   
   - To validate all hosts at the same time, click All in the Node List, then click Validate Host.
To validate more than one host, but not all of them, Ctrl+click the host numbers in the node list, then click Validate Host.

5. Wait while validation proceeds.

The validation step takes several minutes to complete. The tests run in parallel for each host, so the number of hosts does not necessarily increase the amount of time it takes to validate all the hosts if you validate them at the same time. Hosts validation results in one of three possible states:

- Green check mark—The host is valid and can be included in the cluster.

- Orange triangle—The host can be added to the cluster, but warnings were generated. Click the tests in the host validation window to see details about the warnings.

- Red X—The host is not valid. Click the tests in the host validation window that have red X's to see details about the errors. You must correct the errors re-validate or remove the host before MC can create the cluster.

   To remove an invalid host: Highlight the host icon or the IP address in the Node List and click Remove Host.

All hosts must be valid before you can create the cluster. Once all hosts are valid, a Create Cluster button appears near the top right corner of the page.

How to Create the Cluster

1. Click Create Cluster to install Vertica on each host and assemble the nodes into a cluster.

   The process, done in parallel, takes a few minutes as the software is copied to each host and installed.

2. Wait for the process to complete. When the Success dialog opens, you can do one of the following:

   - Optionally create a database on the new cluster at this time by clicking Create Database
   - Click Done to create the database at a later time

See Creating a Database on a Cluster for details on creating a database on the new cluster.
Create a Database on a Cluster

After you use the MC Cluster Installation Wizard to create a Vertica cluster, you can create a database on that cluster through the MC interface. You can create the database on all cluster nodes or on a subset of nodes.

If a database had been created using the Administration Tools on any of the nodes, MC detects (autodiscovers) that database and displays it on the Manage (Cluster Administration) page so you can import it into the MC interface and begin monitoring it.

MC allows only one database running on a cluster at a time, so you might need to stop a running database before you can create a new one.

The following procedure describes how to create a database on a cluster that you created using the MC Cluster Installation Wizard. To create a database on a cluster that you created by running the install_vertica script, see Creating an Empty Database.

Create a Database on a Cluster

To create a new empty database on a new cluster:

1. If you are already on the Databases and Clusters page, skip to the next step. Otherwise:
   a. Connect to MC and sign in as an MC administrator.
   b. On the Home page, click Existing Infrastructure.

2. If no databases exist on the cluster, continue to the next step. Otherwise:
   a. If a database is running on the cluster on which you want to add a new database, select the database and click Stop.
   b. Wait for the running database to have a status of Stopped.

3. Click the cluster on which you want to create the new database and click Create Database.

4. The Create Database wizard opens. Provide the following information:
   - Database name and password. See Creating a Database Name and Password for rules.
   - Optionally click Advanced to open the advanced settings and change the port, and catalog path, and data path. By default the MC application/web server port is 5450 and
paths are /home/dbadmin, or whatever you defined for the paths when you ran the
cluster creation wizard. Do not use the default agent port 5444 as a new setting for the
MC application/web server port. See MC Settings > Configuration for port values.

5. Click Continue.

6. Select nodes to include in the database.

   The Database Configuration window opens with the options you provided and a graphical
   representation of the nodes appears on the page. By default, all nodes are selected to be
   part of this database (denoted by a green check mark). You can optionally click each node
   and clear Include host in new database to exclude that node from the database. Excluded
   nodes are gray. If you change your mind, click the node and select the Include check box.

7. Click Create in the Database Configuration window to create the database on the nodes.

   The creation process takes a few moments and then the database is started and a Success
   message appears.

8. Click OK to close the success message.

   The Database Manager page opens and displays the database nodes. Nodes not included
   in the database are gray.
After You Install Vertica

The tasks described in this section are optional and are provided for your convenience. When you have completed this section, proceed to one of the following:

- **Using This Guide** in Getting Started
- **Configuring the Database** in the Administrator's Guide

Install the License Key

If you did not supply the `-L` parameter during setup, or if you did not bypass the `-L` parameter for a silent install, the first time you log in as the Database Administrator and run the Vertica Administration Tools or Management Console, Vertica requires you to install a license key.

Follow the instructions in **Managing Licenses** in the Administrator's Guide.

Optionally Install vsq1 Client Application on Non-Cluster Hosts

You can use the Vertica vsq1 executable image on a non-cluster Linux host to connect to a Vertica database.

- On Red Hat, CentOS, and SUSE systems, you can install the client driver RPM, which includes the vsq1 executable. See **Installing the Client RPM on Red Hat and SUSE** for details.

- If the non-cluster host is running the same version of Linux as the cluster, copy the image file to the remote system. For example:

  ```
  $ scp host01:/opt/vertica/bin/vsql ./$./vsq1
  ```

- If the non-cluster host is running a different version of Linux than your cluster hosts, and that operating system is not Red Hat version 5 64-bit or SUSE 10/11 64-bit, you must install the Vertica server RPM in order to get vsq1. Download the appropriate rpm package from the Download tab of the myVertica portal then log into the non-cluster host as root and install the rpm package using the command:
In the above command, `filename` is the package you downloaded. Note that you do not have to run the `install_Vertica` script on the non-cluster host in order to use `vsql`.

**Notes**

- Use the same [Command-Line Options](#) that you would on a cluster host.

- You cannot run `vsql` on a Cygwin bash shell (Windows). Use `ssh` to connect to a cluster host, then run `vsql`.

`vsql` is also available for additional platforms. See [Installing the `vsql` Client](#).

**Installing Client Drivers**

After you install Vertica, install drivers on the client systems from which you plan to access your databases. Vertica supplies drivers for ADO.NET, JDBC, ODBC, OLE DB, Perl, and Python. For instructions on installing these drivers, see [Client Drivers](#) in [Connecting to Vertica](#).

**Creating a Database**

To get started using Vertica immediately after installation, create a database. You can use either the Administration Tools or the Management Console. To create a database using MC, refer [Creating a Database Using MC](#).

**Creating a Database Using the Administration Tools**

Follow these step to begin creating a database using the Administration Tools for the first time after installing Vertica.

1. Log in as the database administrator, and type `admintools` to bring up the Administration Tools.
2. When the EULA (end-user license agreement) window opens, type accept to proceed. A window displays, requesting the location of the license key file you downloaded from the Vertica Web site. The default path is /tmp/vlicense.dat.

- If you are using the Vertica Community Edition, click OK without entering a license key.
- If you are using the Vertica Premium Edition, type the absolute path to your license key (for example, /tmp/vlicense.dat) and click OK.

3. From the Administration Tools Main Menu, click Configuration Menu, and then click OK.

4. Click Create Database, and click OK to start the database creation wizard.

For a detailed walkthrough of the database creation wizard for Enterprise Mode and Eon Mode Beta databases, see Creating a Database in the Administrator's Guide.

See Also

- Using the Vertica Interfaces
Upgrading Vertica

The process of upgrading your database with a new Vertica version includes:

- Complete upgrade prerequisites
- Upgrade Vertica
- Perform post-upgrade tasks—required, recommended, and optional

Click on the above links for detailed instructions.

Upgrade Paths

Upgrades are incremental. To upgrade successfully, use the following paths:

- Vertica 7.0 to 7.1
- Vertica 7.1 to 7.2
- Vertica 7.2 to 8.0
- Vertica 8.0 to 8.1
- Vertica 8.1 to 9.0

Note: You cannot upgrade Vertica from versions 7.x to a FIPS-enabled 8.0 system. For more information on FIPS see Federal Information Processing Standard.

Be sure to read the Release Notes and New Features for each version in your path. Documentation for the current Vertica version is available in the RPM, and at http://my.vertica.com/docs, which also provides access to documentation for earlier versions.

To upgrade from an earlier version, please contact Vertica Technical Support for assistance.

Before You Upgrade

Before you upgrade the Vertica database, you must perform the following steps:
- Perform a full database backup. This precautionary measure allows you to restore the current version, if the upgrade is unsuccessful.

- Verify platform requirements for the new version.

- Check catalog storage space.

- Back up any Geospatial indexes and save the results in a temporary table.

After you complete these tasks, shut down the database gracefully. This procedure is described in the Administrator's Guide: Stopping the Database.

Verifying Platform Requirements

The Vertica installer checks the target platform as it runs, and stops whenever it determines the platform fails to meet an installation requirement. Before you update the server package on your systems, manually verify that your platform meets all hardware and software requirements (see Platform Requirements and Recommendations).

By default, the installer stops on all warnings. You can configure the level where the installer stops installation, through the installation parameter --failure-threshold. If you set the failure threshold to FAIL, the installer ignores warnings and stops only on failures.

Caution: Changing the failure threshold lets you immediately upgrade and bring up the Vertica database. However, Vertica cannot fully optimize performance until you correct all warnings.

Checking Catalog Storage Space

Compare how much space the catalog currently uses against space that is available in the same directory:

1. Use the du command to determine how much space the catalog directory now uses:

   $ du -s -BG v_vmart_node0001_catalog
   2G v_vmart_node0001_catalog

2. Determine how much space is available in the same directory:

   $ df -BG v_vmart_node0001_catalog
   Filesystem  1G-blocks  Used  Available  Use%  Mounted on
   /dev/sda2    48G  19G  26G  43%  /
Back Up Geospatial Indexes

When you upgrade, the geospatial indexes are invalid. You must back up spatial indexes that contain polygon geometry data into a temporary table and rebuild them in the newer Place version using the temporary table as the set of input polygons.

The following steps backup your spatial indexes:

1. **Build an index with** `STV_Create_Index`:

   ```sql
   => SELECT STV_Create_Index(123, STGeomFromText('POLYGON((1 2, 2 3, 3 1, 1 2))')
       USING PARAMETERS index='pol_idx' OVER();
   type | polygons | SRID | min_x | min_y | max_x | max_y | info
   +-----------------+---------+-------+-------+-------+-------+-------+-------
   GEOMETRY | 1 | 0 | 1 | 1 | 3 | 3 | (1 row) 
   ```

2. **Save the index polygons in a temporary table. Use** `STV_Describe_Index` **with the** `list_polygons` **option to get the polygons from the index. The \d command describes** `temp-table` **:**

   ```sql
   => CREATE TABLE temp-table
       AS SELECT STV_Create_Index(USING PARAMETERS index='pol_idx', list_polygons=true) OVER();
   => \d temp-table
   Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign Key
   +-----------------+-------+-------+------|------|---------+----------|------------|-----------
   public | temp-table | gid | int | 8 | f | f |
   public | temp-table | state | varchar(20) | 20 | f | f |
   public | temp-table | geometry | geometry(141) | 141 | f | f |
   (3 rows) 
   ```

3. **The temporary table contains a copy of all polygons and identifiers in the indexes. The following command shows the contents of the temporary table:**

   ```sql
   => SELECT gid, state, STAsText(geography) FROM temp-table;
   gid | state | STAsText
   +-----------------+---------+----------------+
   123 | INDEXED | POLYGON ((1 2, 2 3, 3 1, 1 2)) 
   (1 row) 
   ```
After you upgrade, use the polygons table as input to rebuild the index. For details, see Rebuild Geospatial Indexes.

# Upgrade Vertica

**Important:** Before running the upgrade script, be sure to review the tasks described in Before You Upgrade.

Repeat this procedure for each version in your *upgrade path*:

1. Perform a full full *hard-link local backup* of your existing database. This precautionary measure lets you restore from the backup, if the upgrade is unsuccessful. If the upgrade fails, you can reinstall the previous version of Vertica and *restore your database* to that version.

   If your upgrade path includes multiple versions, create a full backup with the first upgrade. For each subsequent upgrade, you can perform incremental backups. However, Vertica recommends full backups before each upgrade if disk space and time allow.

2. Use admintools to *stop the database*.

3. On each host where an additional package is installed, such as the *R language pack*, uninstall it. For example:

   ```sh
   rpm -e vertica-R-lang
   ```

   **Important:** If you omit this step and do not uninstall additional packages, the Vertica server package fails to install in the next step.

4. Make sure you are logged in as root or sudo and use one of the following commands to run the RPM package installer:

   - If you are root and installing an RPM:
     ```sh
     # rpm -Uvh pathname
     ```
   - If you are using sudo and installing an RPM:
     ```sh
     $ sudo rpm -Uvh pathname
     ```
If you are using Debian:

```
$ sudo dpkg -i pathname
```

5. On the same node on which you just installed the RPM, run `update_vertica` as root or sudo. This installs the RPM on all the hosts in the cluster. For example:

**Red Hat or CentOS**

```
#/opt/vertica/sbin/update_vertica --rpm /home/dbadmin/vertica_9.0.x.x86_64.RHEL6.rpm --dba-user mydba
```

**Debian**

```
#/opt/vertica/sbin/update_vertica --deb /home/dbadmin/vertica-amd64.deb --dba-user mydba
```

The following requirements and restrictions apply:

- The DBADMIN user must be able to read the RPM or DEB file when upgrading. Some upgrade scripts are run as the DBADMIN user, and that user must be able to read the RPM or DEB file.

- Use the same options that you used when you last installed or upgraded the database. You can find these options in `/opt/vertica/config/admintools.conf`, on the `install_opts` line. For details on all options, see Installing Vertica with the Installation Script.

  Caution: If you omit any previous options, their default settings are restored. If you do so, or if you change any options, the upgrade script uses the new settings to reconfigure the cluster. This can cause issues with the upgraded database.

- Omit the `--hosts/-h host-list` parameter. The upgrade script automatically identifies cluster hosts.

- If the root user is not in `/etc/sudoers`, an error appears. The installer reports this issue with S0311. See the Sudoers Manual for more information.

6. **Start the database.** The start-up scripts analyze the database and perform necessary data and catalog updates for the new version.

   If Vertica issues a warning stating that one or more packages cannot be installed, run the admintools `--force-reinstall` option to force reinstallation of the packages. For details, see Reinstalling Packages.
7. Perform another database backup.

Post-Upgrade Tasks

After you complete the upgrade, review post-upgrade tasks in After You Upgrade.

After You Upgrade

After you finish upgrading the Vertica server package on your cluster, a number of tasks remain.

Required Tasks

- Upgrade client authentication (required only if you are upgrading from pre-7.1 versions).
- Reinstall packages such as the R language pack that you uninstalled before upgrading. For each package, see its install/upgrade instructions. For R, see Installing/Upgrading the R Language Pack for Vertica.

  Note: Vertica Place is automatically reinstalled with the Vertica server package.

- If the upgrade was unable to install one or more packages, reinstall them with admintools.
- Upgrade Management Console.
- If your Vertica installation is integrated with Hadoop, upgrade the HCatalog connector.

Recommended Tasks

Vertica strongly recommends that you bundle data files after completing the database upgrade.
Optional Tasks

- **Convert backups** so they are compatible with the new version. Database backups from pre-7.2 versions of Vertica are incompatible with the current version.

- Import directed queries that you exported from the previous version. For details, see Batch Query Plan Export and Exporting Directed Queries from the Catalog.

- **Rebuild the geospatial indexes.** For details about the pre-upgrade steps, see Back Up Geospatial Indexes.

Upgrading Client Authentication

**Important:** Perform this task only if you are upgrading from a pre-7.1 version.

Vertica 7.1.0 changed the storage location for the client authentication records from the `vertica.conf` file to the database catalog. When you upgrade from an earlier version, client authentication records in the `vertica.conf` file are converted and inserted into the database catalog. Vertica updates the catalog information on all nodes in the cluster.

Authentication is not enabled after upgrading. As a result, all users can connect to the database. However, if they have a password, they must enter it.

After upgrading, perform the following steps to make sure that client authentication is configured correctly and enabled for use with a running database:

1. Review the client authentication methods that Vertica created during the upgrade by querying the following system tables:

<table>
<thead>
<tr>
<th>System table</th>
<th>Contains information about...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_AUTH</td>
<td>Client authentication methods that Vertica created for your database during the upgrade.</td>
</tr>
<tr>
<td>CLIENT_AUTH_PARAMS</td>
<td>Parameters that Vertica defined for GSS, Ident, and LDAP authentication methods.</td>
</tr>
<tr>
<td>USER_CLIENT_AUTH</td>
<td>An authentication method that you associate with a specific database user through the GRANT (Authentication)</td>
</tr>
</tbody>
</table>
2. Review the vertica.log file to see which authentication records Vertica was not able to create during the upgrade.

3. Create required records as needed with `CREATE AUTHENTICATION`.

4. After the upgrade, enable all defined authentication methods. For each method, enter an `ALTER AUTHENTICATION` statement:

   ```sql
   => ALTER AUTHENTICATION auth-method-name ENABLE;
   ```

5. If you use LDAP over SSL/TLS, you must define the new parameters:

   - `tls_reqcert`
   - `tls_cacert`

   To do so, use `ALTER AUTHENTICATION` as follows:

   ```sql
   => ALTER AUTHENTICATION Ldap1 SET host='ldaps://abc.dc.com',
      binddn_prefix='CN=',
      binddn_suffix='OU=Unit2,DC=dc,DC=com',
      basedn='dc=DC,dc=com',
      tls_cacert='/home/dc.com.ca.cer',
      tls_reqcert='never';
   ```

6. Create an authentication method (LOCAL TRUST or LOCAL PASSWORD) with a very high priority such as 10,000. Grant this method to the DBADMIN user, and set the priority using `ALTER AUTHENTICATION`. For example:

   ```sql
   => CREATE AUTHENTICATION dbadmin_default TRUST LOCAL;
   => ALTER AUTHENTICATION dbadmin_default PRIORITY 10000;
   ```

   With its high priority, this new authentication method supersedes any authentication methods you create for PUBLIC. Even if you make changes to PUBLIC authentication methods, the DBADMIN user can connect to the database at any time.

### Reinstalling Packages

In most cases, Vertica automatically reinstaller all default packages when you restart your database for the first time after running the upgrade script. Occasionally, however, one or more packages might fail to reinstall correctly.
To verify that Vertica succeeded in reinstalling all packages:

1. Restart the database after upgrading.
2. Enter an incorrect password.

If any packages failed to reinstall, Vertica issues a message that specifies the uninstalled packages. In this case, run the admintools command `install_package` with the option `--force-reinstall`:

```
$ admintools -t install_package -d db-name -p password -P pkg-spec --force-reinstall
```

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-d db-name</code></td>
<td>Database name</td>
</tr>
<tr>
<td><code>-p password</code></td>
<td>Database administrator password</td>
</tr>
<tr>
<td><code>-P pkg</code></td>
<td>Specifies which packages to install, where <code>pkg</code> is one of the following:</td>
</tr>
<tr>
<td><code>-package.pkg-spec</code></td>
<td>- The name of a package—for example, <code>flextable</code></td>
</tr>
<tr>
<td></td>
<td>- <code>all</code>: All available packages</td>
</tr>
<tr>
<td></td>
<td>- <code>default</code>: All default packages that are currently installed</td>
</tr>
<tr>
<td><code>--force-reinstall</code></td>
<td>Force installation of a package even if it is already installed.</td>
</tr>
</tbody>
</table>

**Examples**

Force reinstalation of default packages:

```
$ admintools -t install_package -d VMart -p 'password' -P default --force-reinstall
```

Force reinstalation of one package, `flextable`:

```
$ admintools -t install_package -d VMart -p 'password' -P flextable --force-reinstall
```
Upgrading the Advanced Analytics Package 7.2.x to Machine Learning 8.0.x

If you installed the Advanced Analytics package on a 7.2.x Vertica cluster, then you will need to run the following upgrade script to ensure backward comparability with the 9.0.x machine learning functions.

The following command should be run as the dbadmin user:

```
$ vsql -f /opt/vertica/packages/MachineLearning/ddl/install_backward_compatibility.sql
```

Upgrading Management Console

Before You Upgrade

1. Log in as root or a user with sudo privileges on the server where MC is already installed.

2. Open a terminal window and shut down the MC process:

   ```
   # /etc/init.d/vertica-consoled stop
   ```

   For versions of Red Hat 7/CentOS 7 and above, use:

   ```
   # systemctl stop vertica-consoled
   ```

3. Back up MC to preserve configuration metadata.

   **Important:** A full backup is required in order to restore MC to its previous state. Restoring MC is essential if the upgrade fails, or you decide to revert to the previous version of Vertica. For details, see Backing Up MC.

4. Stop the database if the following conditions are true:
   - You are upgrading MC on a Vertica host where Vertica version 7.1.2-5 or below is installed.
   - MC was installed on an Ubuntu or Debian platform.
Tip: If you upgrade from Vertica 7.2.0 or later, the configuration file /opt/vconsole/config/console.properties retains its previous settings. These include the setting for messageCenter.maxEntries, which controls the number of displayed messages. To improve performance, consider setting messageCenter.maxEntries to a value less than 1000.

Extended Monitoring Upgrade Recommendations

If you use Extended Monitoring to monitor a database with MC, Vertica recommends the following upgrade procedure to avoid data loss.

1. Log in to MC as an administrator.

2. To stop the monitored database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click Stop.

3. On MC Settings > MC Storage DB Setup, click Disable Streaming to stop the storage database's collection of monitoring data.

4. To stop the storage database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click Stop.

5. Upgrade MC and Vertica according to Upgrade MC and Upgrading Vertica instructions.

6. To start the storage database, navigate to the Existing Infrastructure > Databases and Clusters page, select the monitored database and click Start.

7. Start the monitored database.

8. On MC Settings > MC Storage DB Setup, click Enable Streaming to enable collection of monitoring data.

To avoid data loss, enable streaming soon after starting your monitored database. While your storage database is down and streaming is disabled, the Kafka server can retain data from your running monitored database for a limited amount of time. Data loss occurs when the data exceeds the Kafka retention policy's log size or retention time limits.
Upgrade MC

1. Download the MC package from the myVertica portal:
   `vertica-console-current-version.Linux-distro`
   Save the package to a location on the target server, such as `/tmp`.

2. On the target server, log in as root or a user with sudo privileges.

3. Change to the directory where you saved the MC package.

4. Install MC using your local Linux distribution package management system—rpm, yum, zypper, apt, dpkg. For example:

   **Red Hat 6**
   
   ```
   # rpm -Uvh vertica-console-current-version.x86_64.RHEL6.rpm
   ```

   **Debian and Ubuntu**
   
   ```
   # dpkg -i vertica-console-current-version.deb
   ```

5. If you stopped the database before upgrading MC, restart the database.

   As the root user, use the following command:

   ```
   /etc/init.d/verticad start
   ```

   **For versions of Red Hat 7/CentOS 7 and above, run:**

   ```
   # systemctl start vertica-consoled
   ```

6. Open a browser and enter the URL of the MC installation, one of the following:

   - **IP address:**
     ```
     https://ip-address:mc-port/
     ```

   - **Server host name:**
     ```
     https://hostname:mc-port/
     ```
By default, \texttt{mc-port} is 5450.

7. If MC was not previously configured, the Configuration Wizard dialog box appears. Configuration steps are described in Configuring MC.

If MC was previously configured, Vertica prompts you to accept the end-user license agreement (EULA) when you first log in to MC after the upgrade.

**Recommended Post-Upgrade Tasks**

After you complete your upgrade, it is strongly recommended that you use the meta-function \texttt{COMPACT_STORAGE} to upgrade existing storage bundles to the .gt format. Bundling reduces the number of files in your file system by at least 50 percent and improves the performance of file-intensive operations. Improved operations include backups, restores, mergeouts, and moveouts.

**Rebuild Geospatial Indexes**

Before you upgrade server versions, you must back up your geospatial indexes in a temporary table. After you upgrade, use \texttt{STV_Create_Index} with the temporary table as input to rebuild the index, and set the \texttt{overwrite} parameter to true to prevent a failure if there is an existing index with the same name:

\[
\begin{verbatim}
=> SELECT STV_Create_Index(gid, geometry USING PARAMETERS index='pol_idx_new', overwrite=true) OVER()
   FROM temp_table;
\end{verbatim}
\]

Now, you can use the index \texttt{pol_idx_new} in the Place version.

**Upgrading the Streaming Data Scheduler Utility**

If you have integrated Vertica with a streaming data application, such as Apache Kafka, you must update the streaming data scheduler utility after you update Vertica.

\textbf{Note:} Schedulers upgraded from version 7.2.x to version 8.0.x are not backwards compatible.

From a command prompt, enter the following command:
/opt/vertica/packages/kafka/bin/vkconfig scheduler --upgrade --upgrade-to-schema schema_name

Running the upgrade task more than once has no effect.

For more information on the Scheduler utility, refer to Scheduler Tool Options.

Upgrading Pre-7.2 Backups

In version 7.2 and later, Vertica no longer relies on hard links to perform backups. As a result, pre-7.2 backups are not compatible with later Vertica versions. To resolve this issue, vbr includes the vbr task 7.2_upgrade. This task copies an existing pre-7.2 backup and creates a 7.2.x-compatible version of it.

**Note:** Vertica recommends that you run this task before performing the first backup of the upgraded database.

To upgrade a backup:

1. Specify the vbr task 7.2_upgrade in the following form:

   ```bash
   vbr -t 7.2_upgrade --old-config-file outdated-configfile.ini -c new-configfile.ini
   ```

2. Verify that the snapshotName parameter is the same in the old and new configuration files.

   The new configuration file assigns new backup locations for the upgraded backup. This approach preserves the existing backup so you can continue to perform incremental backups on the upgraded backup. After the upgrade is complete, Vertica no longer requires the old configuration file.

   If you do not upgrade the backup, the next backup that Vertica executes on the new database is a full backup that subsequently supports incremental backups.
Uninstalling Vertica

For each host in the cluster:

1. Choose a host machine and log in as root (or log in as another user and switch to root).

   
   $ su - root
   password: root-password

2. Find the name of the package that is installed:

   RPM

   # rpm -qa | grep vertica

   DEB

   # dpkg -l | grep vertica

3. Remove the package:

   RPM

   # rpm -e package

   DEB

   # dpkg -r package

Note: If you want to delete the configuration file used with your installation, you can choose to delete the /opt/vertica/ directory and all subdirectories using this command:

   # rm -rf /opt/vertica/
For each client system:

1. Delete the JDBC driver jar file.
2. Delete ODBC driver data source names.
3. Delete the ODBC driver software:
   a. In Windows, go to Start > Control Panel > Add or Remove Programs.
   b. Locate Vertica.
   c. Click Remove.
Uninstalling Management Console

The uninstall command shuts down Management Console and removes most of the files that MC installation script installed.

To uninstall MC:

1. Log in to the target server as root.
2. Stop Management Console:

   ```
   # /etc/init.d/vertica-consoled stop
   ```

   For versions of Red Hat 7/CentOS 7 and above, use:

   ```
   # systemctl stop vertica-consoled
   ```

3. Look for previously-installed versions of MC and note the version:

   **RPM**

   ```
   # rpm -qa | grep vertica
   ```

   **DEB**

   ```
   # dpkg -l | grep vertica
   ```

4. Remove the package:

   **RPM**

   ```
   # rpm -e vertica-console
   ```

   **DEB**

   ```
   # dpkg -r vertica-console
   ```

5. Optionally, delete the MC directory and all subdirectories:

   ```
   # rm -rf /opt/vconsole
   ```
To Reinstall MC

See Installing and Configuring Management Console
Troubleshooting the Vertica Install

The topics described in this section are performed automatically by the install_vertica script and are described in Installing Vertica. If you did not encounter any installation problems, proceed to the Administrator's Guide for instructions on how to configure and operate a database.
Validation Scripts

Vertica provides several validation utilities that can be used prior to deploying Vertica to help determine if your hosts and network can properly handle the processing and network traffic required by Vertica. These utilities can also be used if you are encountering performance issues and need to troubleshoot the issue.

After you install the Vertica RPM, you have access to the following scripts in /opt/vertica/bin:

- **Vcpuperf** - a CPU performance test used to verify your CPU performance.
- **Vioperf** - an Input/Output test used to verify the speed and consistency of your hard drives.
- **Vnetperf** - a Network test used to test the latency and throughput of your network between hosts.

These utilities can be run at any time, but are well suited to use before running the install_vertica script.

**Vcpuperf**

The vcpuperf utility measures your server's CPU processing speed and compares it against benchmarks for common server CPUs. The utility performs a CPU test and measures the time it takes to complete the test. The lower the number scored on the test, the better the performance of the CPU.

The vcpuperf utility also checks the high and low load times to determine if CPU throttling is enabled. If a server's low-load computation time is significantly longer than the high-load computation time, CPU throttling may be enabled. CPU throttling is a power-saving feature. However, CPU throttling can reduce the performance of your server. Vertica recommends disabling CPU throttling to enhance server performance.

**Syntax**

vcpuperf [-q]
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-q</code></td>
<td>Run in quiet mode. Quiet mode displays only the CPU Time, Real Time, and high and low load times.</td>
</tr>
</tbody>
</table>

### Returns

- **CPU Time**: the amount of time it took the CPU to run the test.
- **Real Time**: the total time for the test to execute.
- **High load time**: The amount of time to run the load test while simulating a high CPU load.
- **Low load time**: The amount of time to run the load test while simulating a low CPU load.

### Example

The following example shows a CPU that is running slightly slower than the expected time on a Xeon 5670 CPU that has CPU throttling enabled.

```
[root@node1 bin]# /opt/vertica/bin/vcpuperf
Compiled with: 4.1.2 20080704 (Red Hat 4.1.2-52) Expected time on Core 2, 2.53GHz: ~9.5s
Expected time on Nehalem, 2.67GHz: ~9.0s
Expected time on Xeon 5670, 2.93GHz: ~8.0s

This machine's time:
CPU Time: 8.540000s
Real Time: 8.710000s

Some machines automatically throttle the CPU to save power.
This test can be done in <100 microseconds (60-70 on Xeon 5670, 2.93GHz).
Low load times much larger than 100-200us or much larger than the corresponding high load time indicate low-load throttling, which can adversely affect small query / concurrent performance.

This machine's high load time: 67 microseconds.
This machine's low load time: 208 microseconds.
```
Vioperf

The vioperf utility quickly tests the performance of your host's input and output subsystem. The utility performs the following tests:

- sequential write
- sequential rewrite
- sequential read
- skip read (read non-contiguous data blocks)

The utility verifies that the host reads the same bytes that it wrote and prints its output to STDOUT. The utility also logs the output to a JSON formatted file.

For data in HDFS, the utility tests reads but not writes.

Syntax

```
vioperf [--help] [--duration=INTERVAL] [--log-interval=INTERVAL]
   [<DIR>*]
```

Minimum and Recommended I/O Performance

- The minimum required I/O is 20 MB/s read/write per physical processor core on each node, in full duplex (reading and writing) simultaneously, concurrently on all nodes of the cluster.

- The recommended I/O is 40 MB/s per physical core on each node.

- The minimum required I/O rate for a node with 2 hyper-threaded six-core CPUs (12 physical cores) is 240 MB/s. Vertica recommends 480 MB/s.

For example, the I/O rate for a node with 2 hyper-threaded six-core CPUs (12 physical cores) is 240 MB/s required minimum, 480 MB/s recommended.
## Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--help</strong></td>
<td>Prints a help message and exits.</td>
</tr>
<tr>
<td><strong>--duration</strong></td>
<td>The length of time <code>vioprobe</code> runs performance tests. The default is 5 minutes. Specify the interval in seconds, minutes, or hours with any of these suffixes:</td>
</tr>
<tr>
<td></td>
<td>- Seconds: s, sec, secs, second, seconds. Example: <code>--duration=60sec</code></td>
</tr>
<tr>
<td></td>
<td>- Minutes: m, min, mins, minute, minutes. Example: <code>--duration=10min</code></td>
</tr>
<tr>
<td></td>
<td>- Hours: h, hr, hrs, hour, hours. Example: <code>--duration=1hrs</code></td>
</tr>
<tr>
<td><strong>--log-interval</strong></td>
<td>The interval at which the log file reports summary information. The default interval is 10 seconds. This option uses the same interval notation as <code>--duration</code>.</td>
</tr>
<tr>
<td><strong>--log-file</strong></td>
<td>The path and name where log file contents are written, in JSON. If not specified, then <code>vioperf</code> creates a file named <code>resultsdate-time</code>. JSON in the current directory.</td>
</tr>
<tr>
<td><strong>--condense-log</strong></td>
<td>Directs <code>vioperf</code> to write the log file contents in condensed format, one JSON entry per line, rather than as indented JSON syntax.</td>
</tr>
<tr>
<td><strong>--thread-count=&lt;N&gt;</strong></td>
<td>The number of execution threads to use. By default, <code>vioperf</code> uses all threads available on the host machine.</td>
</tr>
<tr>
<td><strong>--max-buffer-size=&lt;SIZE&gt;</strong></td>
<td>The maximum size of the in-memory buffer to use for reads or writes. Specify the units with any of these suffixes:</td>
</tr>
<tr>
<td></td>
<td>- Bytes: b, byte, bytes.</td>
</tr>
<tr>
<td></td>
<td>- Kilobytes: k, kb, kilobyte, kilobytes.</td>
</tr>
<tr>
<td></td>
<td>- Megabytes: m, mb, megabyte, megabytes.</td>
</tr>
<tr>
<td></td>
<td>- Gigabytes: g, gb, gigabyte, gigabytes.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--preserve-files</td>
<td>Directs vioperf to keep the files it writes. This parameter is ignored for HDFS tests, which are read-only. Inspecting the files can help diagnose write-related failures.</td>
</tr>
<tr>
<td>--disable-crc</td>
<td>Directs vioperf to ignore CRC checksums when validating writes. Verifying checksums can add overhead, particularly when running vioperf on slower processors. This parameter is ignored for HDFS tests.</td>
</tr>
<tr>
<td>--disable-direct-io</td>
<td>When reading from or writing to a local file system, vioperf goes directly to disk by default, bypassing the operating system's page cache. Using direct I/O allows vioperf to measure performance quickly without having to fill the cache. Disabling this behavior can produce more realistic performance results but slows down the operation of vioperf.</td>
</tr>
<tr>
<td>--debug</td>
<td>Directs vioperf to report verbose error messages.</td>
</tr>
<tr>
<td>&lt;DIR&gt;</td>
<td>Zero or more directories to test. If you do not specify a directory, vioperf tests the current directory. To test the performance of each disk, specify different directories mounted on different disks.</td>
</tr>
<tr>
<td></td>
<td>To test reads from a directory on HDFS:</td>
</tr>
<tr>
<td></td>
<td>• Use a URL in the hdfs scheme that points to a single directory (not a path) containing files at least 10MB in size. For best results, use 10GB files and verify that there is at least one file per vioperf thread.</td>
</tr>
<tr>
<td></td>
<td>• If you do not specify a host and port, set the HADOOP_CONF_DIR environment variable to a path including the Hadoop configuration files. This value is the same value that you use for the HadoopConfDir configuration parameter in Vertica. For more information see Configuring the hdfs Scheme.</td>
</tr>
<tr>
<td></td>
<td>• If the HDFS cluster uses Kerberos, set the HADOOP_USER_NAME environment variable to a Kerberos principal.</td>
</tr>
</tbody>
</table>

**Returns**

The utility returns the following information:
<table>
<thead>
<tr>
<th><strong>Heading</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>The test being run (Write, ReWrite, Read, or Skip Read)</td>
</tr>
<tr>
<td>directory</td>
<td>The directory in which the test is being run.</td>
</tr>
<tr>
<td>counter name</td>
<td>The counter type of the test being run. Can be either MB/s or Seeks per second.</td>
</tr>
<tr>
<td>counter value</td>
<td>The value of the counter in MB/s or Seeks per second across all threads. This measurement represents the bandwidth at the exact time of measurement. Contrast with counter value (avg).</td>
</tr>
<tr>
<td>counter value (10 sec avg)</td>
<td>The average amount of data in MB/s, or the average number of Seeks per second, for the test being run in the duration specified with --log-interval. The default interval is 10 seconds. The counter value (avg) is the average bandwidth since the last log message, across all threads.</td>
</tr>
<tr>
<td>counter value/core</td>
<td>The counter value divided by the number of cores.</td>
</tr>
<tr>
<td>counter value/core (10 sec avg)</td>
<td>The counter value (10 sec avg) divided by the number of cores.</td>
</tr>
<tr>
<td>thread count</td>
<td>The number of threads used to run the test.</td>
</tr>
<tr>
<td>%CPU</td>
<td>The available CPU percentage used during this test.</td>
</tr>
<tr>
<td>%I/O Wait</td>
<td>The CPU percentage in I/O Wait state during this test. I/O wait state is the time working processes are blocked while waiting for I/O operations to complete.</td>
</tr>
<tr>
<td>elapsed time</td>
<td>The amount of time taken for a particular test. If you run the test multiple times, elapsed time increases the next time the test is run.</td>
</tr>
<tr>
<td>remaining time</td>
<td>The time remaining until the next test. Based on the --duration option, each of the tests is run at least once. If the test set is run multiple times, then remaining time is how much longer the test will run. The remaining time value is cumulative. Its total is added to elapsed time each time the same test is run again.</td>
</tr>
</tbody>
</table>
Example

Invoking `vioperf` from a terminal outputs the following message and sample results:

```
[dbadmin@v_mart_node0001 ~]$ /opt/vertica/bin/vioperf --duration=60s
The minimum required I/O is 20 MB/s read and write per physical processor core on each node, in full
duplex
i.e. reading and writing at this rate simultaneously, concurrently on all nodes of the cluster. The
recommended I/O is 40 MB/s per physical core on each node.
For example, the I/O rate for a server node with 2 hyper-threaded six-core CPUs is 240 MB/s required
minimum, 480 MB/s recommended.
Using direct io (buffer size=1048576, alignment=512) for directory "/home/dbadmin"
```

```
test  | directory   | counter name | counter value | counter value (10 sec avg) | counter value/core | counter value/core (10 sec avg) | thread count | %CPU | %IO Wait | elapsed time (s) | remaining time (s)
-----------------------------------------------
-----------------------------------------------
Write | /home/dbadmin | MB/s         | 420           | 420                         |                     |                            | 2            | 89   | 10      | 10                 |
210   | 210           | 2            | 89            | 10                          |                      |                            |              |      | 10      |                    |
| 5     | 5             |              |                |                             |                      |                            |              |      |          |                    |
| 198   | 198           | 2            | 89            | 9                           |                      |                            |              |      | 15      |                    |
Write | /home/dbadmin | MB/s         | 412           | 396                         |                     |                            | 2            | 89   | 9       | 15                 |
286   | 286           | 2            | 89            | 9                           |                      |                            |              |      | 15      |                    |
| 0     | 0             |              |                |                             |                      |                            |              |      |          |                    |
| 75+75 | 75+75         | 2            | 58            | 40                          | 10                  |                            |              |      |          |                    |
ReWrite| /home/dbadmin | (MB-read+MB-write)/s | 150+150       | 150+150                     |                     |                            |              |      |          |                    |
79+79  | 79+79         | 2            | 58            | 40                          | 10                  |                            |              |      |          |                    |
| 5     | 5             |              |                |                             |                      |                            |              |      |          |                    |
| 86+86 | 86+86         | 2            | 64            | 33                          | 15                  |                            |              |      |          |                    |
| 0     | 0             |              |                |                             |                      |                            |              |      |          |                    |
| 97    | 97            | 2            | 69            | 26                          | 10                  |                            |              |      |          |                    |
| 5     | 5             |              |                |                             |                      |                            |              |      |          |                    |
| 95    | 95            | 2            | 71            | 27                          | 15                  |                            |              |      |          |                    |
| 0     | 0             |              |                |                             |                      |                            |              |      |          |                    |
Read  | /home/dbadmin | MB/s         | 194           | 194                         |                     |                            | 2            | 69   | 26      | 10                 |
96    | 96            | 2            | 69            | 26                          | 10                  |                            |              |      |          |                    |
| 0     | 0             |              |                |                             |                      |                            |              |      |          |                    |
| 329.5 | 329.5         | 2            | 85            | 10                          |                      |                            |              |      |          |                    |
| 5     | 5             |              |                |                             |                      |                            |              |      |          |                    |
| 357   | 357           | 2            | 59            | 15                          |                      |                            |              |      |          |                    |
| 0     | 0             |              |                |                             |                      |                            |              |      |          |                    |
```

Note: When evaluating performance for minimum and recommended I/O, include the
Write and Read values in your evaluation. ReWrite and SkipRead values are not relevant to
determining minimum and recommended I/O.
Vnetperf

The vnetperf utility allows you to measure the network performance of your hosts. It can measure network latency and the throughput for both the TCP and UDP protocols.

**Important:** This utility introduces a high network load and must not be used on a running Vertica cluster or database performance is degraded.

Using this utility you can detect:

- if throughput is low for all hosts or a particular host,
- if latency is high for all hosts or a particular host,
- bottlenecks between one or more hosts or subnets,
- too low a limit in the number of TCP connections that can be established simultaneously,
- and if there is a high rate of packet loss on the network.

The latency test measures the latency from the host running the script to the other hosts. Any host that has a particularly high latency should be investigated further.

The throughput tests measure both UDP and TCP throughput. You can specify a rate limit in MB/s to use for these tests, or allow the utility to use a range of throughputs to be used.

**Syntax**

```
 vnetperf [options] [tests]
```

**Recommended Network Performance**

- The maximum recommended RTT (round-trip time) latency is 1000 microseconds. The ideal RTT latency is 200 microseconds or less. Vertica recommends that clock skew be kept to under 1 second.

- The minimum recommended throughput is 100 MB/s. Ideal throughput is 800 MB/s or more.
Note: UDP numbers may be lower, multiple network switches may reduce performance results.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--condense</td>
<td>Condense the log into one JSON entry per line, instead of indented JSON syntax.</td>
</tr>
<tr>
<td>--collect-logs</td>
<td>Collect the test log files from each host.</td>
</tr>
<tr>
<td>--datarate rate</td>
<td>Limit the throughput to this rate in MB/s. A rate of 0 loops the tests through several different rates. The default is 0.</td>
</tr>
<tr>
<td>--duration seconds</td>
<td>The time limit for each test to run in seconds. The default is 1.</td>
</tr>
<tr>
<td>--hosts host1,host2,...</td>
<td>A comma-separated list of hosts on which to run the tests. Do not use spaces between the comma's and the host names.</td>
</tr>
<tr>
<td>--hosts file</td>
<td>A hosts file that specifies the hosts on which to run the tests. If the --hosts argument is not used, then the utility attempts to access admintools and determine the hosts in the cluster.</td>
</tr>
<tr>
<td>--identity-file file</td>
<td>If using passwordless SSH/SCP access between the hosts, then specify the key file used to gain access to the hosts.</td>
</tr>
<tr>
<td>--ignore-bad-hosts</td>
<td>If set, run the tests on the reachable hosts even if some hosts are not reachable. If not set, and a host is unreachable, then no tests are run on any hosts.</td>
</tr>
<tr>
<td>--log-dir directory</td>
<td>If --collect-logs is set, the directory in which to place the collected logs. The default directory is named logs.netperf.&lt;timestamp&gt;</td>
</tr>
<tr>
<td>--log-level LEVEL</td>
<td>The log level to use. Possible values are: INFO, ERROR, DEBUG, and WARN. The default is WARN.</td>
</tr>
<tr>
<td>--list-tests</td>
<td>Lists the tests that can be run by this utility.</td>
</tr>
<tr>
<td>--output-file file</td>
<td>The file that JSON results are written to. The default is results.&lt;timestamp&gt;.json.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>--ports port1, port2, port3</td>
<td>The port numbers to use. If only one is specified then the next two numbers in sequence are also used. The default ports are 14159, 14160, 14161.</td>
</tr>
<tr>
<td>--scp-options 'options'</td>
<td>Using this argument, you can specify one or more standard SCP command line arguments enclosed in single quotes. SCP is used to copy test binaries over to the target hosts.</td>
</tr>
<tr>
<td>--ssh-options 'options'</td>
<td>Using this argument, you can specify one or more standard SSH command line arguments enclose in single quotes. SSH is used to issue test commands on the target hosts.</td>
</tr>
<tr>
<td>--vertica-install directory</td>
<td>If specified, then the utility assumes Vertica is installed on each of the hosts and to use the test binaries on the target system rather than copying them over using SCP.</td>
</tr>
</tbody>
</table>

**Tests**

Note: If the tests argument is omitted then all tests are run.

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>Test the latency to each of the hosts.</td>
</tr>
<tr>
<td>tcp-throughput</td>
<td>Test the TCP throughput amongst the hosts.</td>
</tr>
<tr>
<td>udp-throughput</td>
<td>Test the UDP throughput amongst the hosts.</td>
</tr>
</tbody>
</table>

**Returns**

For each host it returns the following:

Latency test returns:

- The Round Trip Time (rtt) latency for each host in milliseconds.
- Clock Skew = the difference in time shown by the clock on the target host relative to the host running the utility.

UDP and TCP throughput tests return:
- The date/time and test name.
- The rate limit in MB/s.
- The node being tested.
- Sent and Received data in MB/s and bytes.
- The duration of the test in seconds.

Example

```
/opt/vertica/bin/vnetperf --condense -hosts 10.20.100.66,10.20.100.67 --identity-file '/root/.ssh/vid_rsa'
```

Enable Secure Shell (SSH) Logins

The administrative account must be able to use Secure Shell (SSH) to log in (ssh) to all hosts without specifying a password. The shell script `install_vertica` does this automatically. This section describes how to do it manually if necessary.

1. If you do not already have SSH installed on all hosts, log in as root on each host and install it now. You can download a free version of the SSH connectivity tools from OpenSSH.

2. Log in to the Vertica administrator account (dbadmin in this example).

3. Make your home directory (~) writable only by yourself. Choose one of:

```
$ chmod 700 ~
```

or

```
$ chmod 755 ~
```

where:

<table>
<thead>
<tr>
<th>700 includes</th>
<th>755 includes</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 read by owner</td>
<td>400 read by owner</td>
</tr>
</tbody>
</table>
4. Change to your home directory:

$ cd ~

5. Generate a private key/ public key pair:

$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/dbadmin/.ssh/id_rsa):
Created directory '/home/dbadmin/.ssh'.
Enter passphrase (empty for no passphrase): 
Enter same passphrase again: 
Your identification has been saved in /home/dbadmin/.ssh/id_rsa.
Your public key has been saved in /home/dbadmin/.ssh/id_rsa.pub.

6. Make your .ssh directory readable and writable only by yourself:

$ chmod 700 ~/.ssh

7. Change to the .ssh directory:

$ cd ~/.ssh

8. Copy the file id_rsa.pub onto the file authorized_keys2.

$ cp id_rsa.pub authorized_keys2

9. Make the files in your .ssh directory readable and writable only by yourself:

$ chmod 600 ~/.ssh/*
For each cluster host:

```
$ scp -r ~/.ssh <host>:. 
```

Connect to each cluster host. The first time you ssh to a new remote machine, you could get a message similar to the following:

```
$ ssh dev0 Warning: Permanently added 'dev0,192.168.1.92' (RSA) to the list of known hosts.
```

This message appears only the first time you ssh to a particular remote host.

See Also

- OpenSSH
Upgrading Your Operating System on Nodes in Your Vertica Cluster

If you need to upgrade the operating system on the nodes in your Vertica cluster, check with the documentation for your Linux distribution to make sure they support the particular upgrade you are planning.

For example, the following articles provide information about upgrading Red Hat:

- How do I upgrade from Red Hat Enterprise Linux 6 to Red Hat Enterprise Linux 7?
- Does Red Hat support upgrades between major versions of Red Hat Enterprise Linux?

After you confirm that you can perform the upgrade, follow the steps at Best Practices for Upgrading the Operating System on Nodes in a Vertica Cluster.
Upgrading the Operating System on the Nodes in Your Vertica Cluster

For information about upgrading the operating system on the nodes in your Vertica cluster, see Best Practices for Upgrading the Operating System on Nodes in a Vertica Cluster.
Appendix: Time Zones

- Using Time Zones With Vertica
- Africa
- America
- Antarctica
- Asia
- Atlantic
- Australia
- Etc/GMT
- Europe
- Indian
- Pacific

Using Time Zones With Vertica

Vertica uses the TZ environment variable on each node, if it has been set, for the default current time zone. Otherwise, Vertica uses the operating system time zone.

The TZ variable can be set by the operating system during login (see /etc/profile, /etc/profile.d, or /etc/bashrc) or by the user in .profile, .bashrc or .bash-profile.

TZ must be set to the same value on each node when you start Vertica.

The following command returns the current time zone for your database:

```
=> SHOW TIMEZONE;
    name | setting
---------------------
timezone | America/New_York
(1 row)
```
You can also use the `SET TIMEZONE TO { value | 'value' }` command to set the time zone for a single session.

There is no database default time zone; instead, `TIMESTAMP WITH TIMEZONE (TIMESTAMPTZ)` data is stored in GMT (UTC) by converting data from the current local time zone to GMT.

When TIMESTAMPTZ data is used, data is converted back to use the current local time zone, which might be different from the local time zone where the data was stored. This conversion takes into account Daylight Saving Time (Summer Time), if applicable, depending on the year and date, to know when the Daylight Saving Time change occurred.

TIMESTAMP WITHOUT TIMEZONE data stores the timestamp, as given, and retrieves it exactly as given. The current time zone is ignored. The same is true for TIME WITHOUT TIMEZONE. For TIME WITH TIMEZONE (TIMETZ), however, the current time zone setting is stored along with the given time, and that time zone is used on retrieval.

**Note:** Vertica recommends that you use TIMESTAMPTZ, not TIMETZ.

TIMESTAMPTZ uses the current time zone on both input and output, such as in the following example:

```sql
=> CREATE TEMP TABLE s (tstz TIMESTAMPTZ);=> SET TIMEZONE TO 'America/New_York';
=> INSERT INTO s VALUES ('2009-02-01 00:00:00');
=> INSERT INTO s VALUES ('2009-05-12 12:00:00');
=> SELECT tstz AS 'Local timezone', tstz AT TIMEZONE 'America/New_York' AS 'America/New_York',
   tstz AT TIMEZONE 'GMT' AS 'GMT' FROM s;

<table>
<thead>
<tr>
<th>Local timezone</th>
<th>America/New_York</th>
<th>GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-01 00:00:00-05</td>
<td>2009-02-01 00:00:00</td>
<td>2009-02-01 05:00:00</td>
</tr>
<tr>
<td>2009-05-12 12:00:00-04</td>
<td>2009-05-12 12:00:00</td>
<td>2009-05-12 16:00:00</td>
</tr>
</tbody>
</table>
(2 rows)
```

The -05 in the Local time zone column above shows that the data is displayed in EST, while -04 indicates EDT. The other two columns show the TIMESTAMP WITHOUT TIMEZONE at the specified time zone.

The next example illustrates what occurs if the current time zone is changed to, for example, Greenwich Mean Time:

```sql
=> SET TIMEZONE TO 'GMT';=> SELECT tstz AS 'Local timezone', tstz AT TIMEZONE 'America/New_York' AS 'America/New_York', tstz AT TIMEZONE 'GMT' AS 'GMT' FROM s;

<table>
<thead>
<tr>
<th>Local timezone</th>
<th>America/New_York</th>
<th>GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-01 05:00:00+00</td>
<td>2009-02-01 00:00:00</td>
<td>2009-02-01 05:00:00</td>
</tr>
<tr>
<td>2009-05-12 16:00:00+00</td>
<td>2009-05-12 12:00:00</td>
<td>2009-05-12 16:00:00</td>
</tr>
</tbody>
</table>
(2 rows)
```

The +00 in the Local time zone column above indicates that TIMESTAMPTZ is displayed in 'GMT'.
The approach of using TIMESTAMPTZ fields to record events captures the GMT of the event, as expressed in terms of the local time zone. Later, it allows for easy conversion to any other time zone, either by setting the local time zone or by specifying an explicit AT TIMEZONE clause.

The following example shows how TIMESTAMP WITHOUT TIMEZONE fields work in Vertica.

```sql
=> CREATE TEMP TABLE tnoz (ts TIMESTAMP);
=> INSERT INTO tnoz VALUES('2009-02-01 00:00:00');
=> INSERT INTO tnoz VALUES('2009-05-12 12:00:00');
=> SET TIMEZONE TO 'GMT';
=> SELECT ts AS 'No timezone', ts AT TIMEZONE 'America/New_York' AS 'America/New_York', ts AT TIMEZONE 'GMT' AS 'GMT' FROM tnoz;
```

<table>
<thead>
<tr>
<th></th>
<th>America/New_York</th>
<th>GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-01 00:00:00</td>
<td>2009-02-01 05:00:00+00</td>
<td>2009-02-01 00:00:00+00</td>
</tr>
<tr>
<td>2009-05-12 12:00:00</td>
<td>2009-05-12 16:00:00+00</td>
<td>2009-05-12 12:00:00+00</td>
</tr>
</tbody>
</table>

(2 rows)

The +00 at the end of a timestamp indicates that the setting is TIMESTAMP WITH TIMEZONE in GMT (the current time zone). The 'America/New_York' column shows what the 'GMT' setting was when you recorded the time, assuming you read a normal clock in the time zone 'America/New_York'. What this shows is that if it is midnight in the 'America/New_York' time zone, then it is 5 am GMT.

**Note:** 00:00:00 Sunday February 1, 2009 in America/New_York converts to 05:00:00 Sunday February 1, 2009 in GMT.

The 'GMT' column displays the GMT time, assuming the input data was captured in GMT.

If you don't set the time zone to GMT, and you use another time zone, for example 'America/New_York', then the results display in 'America/New_York' with a -05 and -04, showing the difference between that time zone and GMT.

```sql
=> SET TIMEZONE TO 'America/New_York';
=> SHOW TIMEZONE;
  name | setting
----------+----------
 timezon  | America/New_York
----------+----------
(1 row)
=> SELECT ts AS 'No timezone', ts AT TIMEZONE 'America/New_York' AS 'America/New_York', ts AT TIMEZONE 'GMT' AS 'GMT' FROM tnoz;
```

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>America/New_York</th>
<th>GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-01 00:00:00</td>
<td>2009-02-01 00:00:00-05</td>
<td>2009-01-31 19:00:00-05</td>
</tr>
<tr>
<td>2009-05-12 12:00:00</td>
<td>2009-05-12 12:00:00-04</td>
<td>2009-05-12 08:00:00-04</td>
</tr>
</tbody>
</table>
```

(2 rows)

In this case, the last column is interesting in that it returns the time in New York, given that the data was captured in 'GMT'.

Vertica Analytic Database (9.0.x)  Page 249 of 6180
## See Also

- TZ Environment Variable
- SET TIME ZONE
- Date/Time Data Types

### Africa

<table>
<thead>
<tr>
<th>Continent</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Africa</td>
<td>Abidjan</td>
</tr>
<tr>
<td></td>
<td>Accra</td>
</tr>
<tr>
<td></td>
<td>Addis_Ababa</td>
</tr>
<tr>
<td></td>
<td>Algiers</td>
</tr>
<tr>
<td></td>
<td>Asmera</td>
</tr>
<tr>
<td></td>
<td>Bamako</td>
</tr>
<tr>
<td></td>
<td>Bangui</td>
</tr>
<tr>
<td></td>
<td>Banjul</td>
</tr>
<tr>
<td></td>
<td>Bissau</td>
</tr>
<tr>
<td></td>
<td>Blantyre</td>
</tr>
<tr>
<td></td>
<td>Brazzaville</td>
</tr>
<tr>
<td></td>
<td>Bujumbura</td>
</tr>
<tr>
<td></td>
<td>Cairo_Egypt</td>
</tr>
<tr>
<td></td>
<td>Casablanca</td>
</tr>
<tr>
<td></td>
<td>Ceuta</td>
</tr>
<tr>
<td></td>
<td>Conakry</td>
</tr>
</tbody>
</table>
### America

<table>
<thead>
<tr>
<th>America/Adak</th>
</tr>
</thead>
<tbody>
<tr>
<td>America/Atka</td>
</tr>
<tr>
<td>America/Aleutian</td>
</tr>
<tr>
<td>America/Atka US/Aleutian</td>
</tr>
<tr>
<td>America/Atka US/Aleutia</td>
</tr>
<tr>
<td>America/Anchorage</td>
</tr>
<tr>
<td>America/Anchorage SystemV/YST9YDT</td>
</tr>
<tr>
<td>America/Anchorage SystemV/YST9YDT US/Alaska</td>
</tr>
<tr>
<td>America/Anguilla</td>
</tr>
<tr>
<td>America/Antigua</td>
</tr>
<tr>
<td>America/Araguaina</td>
</tr>
<tr>
<td>America/Aruba</td>
</tr>
<tr>
<td>America/Asuncion</td>
</tr>
<tr>
<td>America/Bahia</td>
</tr>
<tr>
<td>America/Barbados</td>
</tr>
<tr>
<td>Time Zone</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>America/Belem</td>
</tr>
<tr>
<td>America/Belize</td>
</tr>
<tr>
<td>America/Boa_Vista</td>
</tr>
<tr>
<td>America/Bogota</td>
</tr>
<tr>
<td>America/Boise</td>
</tr>
<tr>
<td>America/Buenos_Aires</td>
</tr>
<tr>
<td>America/Cambridge_Bay</td>
</tr>
<tr>
<td>America/Campo_Grande</td>
</tr>
<tr>
<td>America/Cancun</td>
</tr>
<tr>
<td>America/Caracas</td>
</tr>
<tr>
<td>America/Catamarca</td>
</tr>
<tr>
<td>America/Cayenne</td>
</tr>
<tr>
<td>America/Cayman</td>
</tr>
<tr>
<td>America/Chicago CST6CDT SystemV/CST6CDT US/Central</td>
</tr>
<tr>
<td>America/Chihuahua</td>
</tr>
<tr>
<td>America/Cordoba America/Rosario</td>
</tr>
<tr>
<td>America/Costa_Rica</td>
</tr>
<tr>
<td>America/Cuiaba</td>
</tr>
<tr>
<td>America/Curacao</td>
</tr>
<tr>
<td>America/Danmarkshavn</td>
</tr>
<tr>
<td>America/Dawson</td>
</tr>
<tr>
<td>America/Dawson_Creek</td>
</tr>
<tr>
<td>America/Denver MST7MDT SystemV/MST7MDT US/Mountain America/Shiprock Navajo</td>
</tr>
<tr>
<td>America/Detroit US/Michigan</td>
</tr>
<tr>
<td>Time Zone</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>America/Dominica</td>
</tr>
<tr>
<td>America/Edmonton Canada/Mountain</td>
</tr>
<tr>
<td>America/Eirunepe</td>
</tr>
<tr>
<td>America/El_Salvador</td>
</tr>
<tr>
<td>America/Ensenada America/Tijuana Mexico/BajaNorte</td>
</tr>
<tr>
<td>America/Fortaleza</td>
</tr>
<tr>
<td>America/Glace_Bay</td>
</tr>
<tr>
<td>America/Godthab</td>
</tr>
<tr>
<td>America/Goose_Bay</td>
</tr>
<tr>
<td>America/Grand_Turk</td>
</tr>
<tr>
<td>America/Grenada</td>
</tr>
<tr>
<td>America/Guadeloupe</td>
</tr>
<tr>
<td>America/Guatemala</td>
</tr>
<tr>
<td>America/Guayaquil</td>
</tr>
<tr>
<td>America/Guyana</td>
</tr>
<tr>
<td>America/Halifax Canada/Atlantic SystemV/AST4ADT</td>
</tr>
<tr>
<td>America/Havana Cuba</td>
</tr>
<tr>
<td>America/Hermosillo</td>
</tr>
<tr>
<td>America/Indiana/Indianapolis</td>
</tr>
<tr>
<td>America/Indianapolis</td>
</tr>
<tr>
<td>America/Fort_Wayne EST SystemV/EST5 US/East-Indiana</td>
</tr>
<tr>
<td>America/Indiana/Knox America/Knox_IN US/Indiana-Starke</td>
</tr>
<tr>
<td>America/Indiana/Marengo</td>
</tr>
<tr>
<td>America/Indiana/Vevay</td>
</tr>
<tr>
<td>America/Inuvik</td>
</tr>
<tr>
<td>Time Zone</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>America/Iqaluit</td>
</tr>
<tr>
<td>America/Jamaica</td>
</tr>
<tr>
<td>America/Jujuy</td>
</tr>
<tr>
<td>America/Juneau</td>
</tr>
<tr>
<td>America/Kentucky/Louisville</td>
</tr>
<tr>
<td>America/Kentucky/Monticello</td>
</tr>
<tr>
<td>America/La_Paz</td>
</tr>
<tr>
<td>America/Lima</td>
</tr>
<tr>
<td>America/Los_Angeles</td>
</tr>
<tr>
<td>America/Maceio</td>
</tr>
<tr>
<td>America/Managua</td>
</tr>
<tr>
<td>America/Manaus Brazil/West</td>
</tr>
<tr>
<td>America/Martinique</td>
</tr>
<tr>
<td>America/Mazatlan Mexico/BajaSur</td>
</tr>
<tr>
<td>America/Mendoza</td>
</tr>
<tr>
<td>America/Menominee</td>
</tr>
<tr>
<td>America/Merida</td>
</tr>
<tr>
<td>America/Mexico_City Mexico/General</td>
</tr>
<tr>
<td>America/Miquelon</td>
</tr>
<tr>
<td>America/Monterrey</td>
</tr>
<tr>
<td>America/Montevideo</td>
</tr>
<tr>
<td>America/Montreal</td>
</tr>
<tr>
<td>America/Montserrat</td>
</tr>
<tr>
<td>America/Nassau</td>
</tr>
<tr>
<td>Time Zone</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>America/New_York</td>
</tr>
<tr>
<td>America/Nipigon</td>
</tr>
<tr>
<td>America/Nome</td>
</tr>
<tr>
<td>America/Noronha</td>
</tr>
<tr>
<td>America/North_Dakota</td>
</tr>
<tr>
<td>America/Panama</td>
</tr>
<tr>
<td>America/Pangnirtung</td>
</tr>
<tr>
<td>America/Paramaribo</td>
</tr>
<tr>
<td>America/Phoenix</td>
</tr>
<tr>
<td>America/Port-au-Prince</td>
</tr>
<tr>
<td>America/Port_of_Spain</td>
</tr>
<tr>
<td>America/Porto_Acre</td>
</tr>
<tr>
<td>America/Rio_Branco</td>
</tr>
<tr>
<td>America/Porto_Velho</td>
</tr>
<tr>
<td>America/Puerto_Rico</td>
</tr>
<tr>
<td>America/Rainy_River</td>
</tr>
<tr>
<td>America/Recife</td>
</tr>
<tr>
<td>America/Regina</td>
</tr>
<tr>
<td>America/Santiago</td>
</tr>
<tr>
<td>America/Santo_Doming</td>
</tr>
<tr>
<td>America/Sao_Paulo</td>
</tr>
<tr>
<td>America/Scoresbysund</td>
</tr>
<tr>
<td>America/St_Johns</td>
</tr>
<tr>
<td>America/St_Kitts</td>
</tr>
</tbody>
</table>
### America
- America/St_Lucia
- America/St_Thomas America/Virgin
- America/St_Vincent
- America/Swift_Current
- America/Tegucigalpa
- America/Thule
- America/Thunder_Bay
- America/Toronto Canada/Eastern
- America/Tortola
- America/Vancouver Canada/Pacific
- America/Whitehorse Canada/Yukon
- America/Winnipeg Canada/Central
- America/Yakutat
- America/Yellowknife

### Antarctica
- Antarctica/Casey
- Antarctica/Davis
- Antarctica/DumontDUrville
- Antarctica/Mawson
- Antarctica/McMurdo
- Antarctica/South_Pole
- Antarctica/Palmer
Asia

Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Choibalsan
Asia/Chongqing Asia/Chungking
Asia/Colombo
<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia/Makassar Asia/Ujung_Pandang</td>
</tr>
<tr>
<td>Asia/Manila</td>
</tr>
<tr>
<td>Asia/Muscat</td>
</tr>
<tr>
<td>Asia/Nicosia Europe/Nicosia</td>
</tr>
<tr>
<td>Asia/Novosibirsk</td>
</tr>
<tr>
<td>Asia/Omsk</td>
</tr>
<tr>
<td>Asia/Oral</td>
</tr>
<tr>
<td>Asia/Phnom_Penh</td>
</tr>
<tr>
<td>Asia/Pontianak</td>
</tr>
<tr>
<td>Asia/Pyongyang</td>
</tr>
<tr>
<td>Asia/Qatar</td>
</tr>
<tr>
<td>Asia/Qyzylorda</td>
</tr>
<tr>
<td>Asia/Rangoon</td>
</tr>
<tr>
<td>Asia/Riyadh</td>
</tr>
<tr>
<td>Asia/Riyadh87 Mideast/Riyadh87</td>
</tr>
<tr>
<td>Asia/Riyadh88 Mideast/Riyadh88</td>
</tr>
<tr>
<td>Asia/Riyadh89 Mideast/Riyadh89</td>
</tr>
<tr>
<td>Asia/Saigon</td>
</tr>
<tr>
<td>Asia/Sakhalin</td>
</tr>
<tr>
<td>Asia/Samarkand</td>
</tr>
<tr>
<td>Asia/Seoul ROK</td>
</tr>
<tr>
<td>Asia/Shanghai PRC</td>
</tr>
<tr>
<td>Asia/Singapore Singapore</td>
</tr>
<tr>
<td>Asia/Taipei ROC</td>
</tr>
<tr>
<td>Asian Time Zones</td>
</tr>
<tr>
<td>------------------------------------</td>
</tr>
<tr>
<td>Asia/Tashkent</td>
</tr>
<tr>
<td>Asia/Tbilisi</td>
</tr>
<tr>
<td>Asia/Tehran Iran</td>
</tr>
<tr>
<td>Asia/Thimbu Asia/Thimpu</td>
</tr>
<tr>
<td>Asia/Tokyo Japan</td>
</tr>
<tr>
<td>Asia/Ulaanbaatar Asia/Ulaan_Bator</td>
</tr>
<tr>
<td>Asia/Urumqi</td>
</tr>
<tr>
<td>Asia/Vientiane</td>
</tr>
<tr>
<td>Asia/Vladivostok</td>
</tr>
<tr>
<td>Asia/Yakutsk</td>
</tr>
<tr>
<td>Asia/Yekaterinburg</td>
</tr>
<tr>
<td>Asia/Yerevan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atlantic Time Zones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlantic/Azores</td>
</tr>
<tr>
<td>Atlantic/Bermuda</td>
</tr>
<tr>
<td>Atlantic/Canary</td>
</tr>
<tr>
<td>Atlantic/Cape_Verde</td>
</tr>
<tr>
<td>Atlantic/Faeroe</td>
</tr>
<tr>
<td>Atlantic/Madeira</td>
</tr>
<tr>
<td>Atlantic/Reykjavik Iceland</td>
</tr>
<tr>
<td>Atlantic/South_Georgia</td>
</tr>
<tr>
<td>Atlantic/St_Helena</td>
</tr>
<tr>
<td>Atlantic/Stanley</td>
</tr>
</tbody>
</table>
## Australia

<table>
<thead>
<tr>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia/ACT</td>
</tr>
<tr>
<td>Australia/Canberra</td>
</tr>
<tr>
<td>Australia/NSW</td>
</tr>
<tr>
<td>Australia/Sydney</td>
</tr>
<tr>
<td>Australia/Adelaide</td>
</tr>
<tr>
<td>Australia/South</td>
</tr>
<tr>
<td>Australia/Brisbane</td>
</tr>
<tr>
<td>Australia/Queensland</td>
</tr>
<tr>
<td>Australia/Broken_Hill</td>
</tr>
<tr>
<td>Australia/Yancowinna</td>
</tr>
<tr>
<td>Australia/Darwin</td>
</tr>
<tr>
<td>Australia/North</td>
</tr>
<tr>
<td>Australia/Hobart</td>
</tr>
<tr>
<td>Australia/Tasmania</td>
</tr>
<tr>
<td>Australia/LHI</td>
</tr>
<tr>
<td>Australia/Lord_Howe</td>
</tr>
<tr>
<td>Australia/Lindeman</td>
</tr>
<tr>
<td>Australia/Melbourne</td>
</tr>
<tr>
<td>Australia/Victoria</td>
</tr>
<tr>
<td>Australia/Perth</td>
</tr>
<tr>
<td>Australia/West</td>
</tr>
</tbody>
</table>
**Etc/GMT**

- Etc/GMT+0...Etc/GMT+12
- Etc/GMT-0...ETC/GMT-14

**Europe**

- Europe/Amsterdam
- Europe/Andorra
- Europe/Athens
- Europe/Belfast
- Europe/Belgrade
- Europe/Ljubljana
- Europe/Sarajevo
- Europe/Skopje
- Europe/Zagreb
- Europe/Berlin
- Europe/Brussels
- Europe/Bucharest
- Europe/Budapest
- Europe/Chisinau
- Europe/Tiraspol
- Europe/Copenhagen
- Europe/Dublin Eire
- Europe/Gibraltar
<table>
<thead>
<tr>
<th>Time Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe/Helsinki</td>
</tr>
<tr>
<td>Europe/Istanbul Asia/Istanbul Turkey</td>
</tr>
<tr>
<td>Europe/Kaliningrad</td>
</tr>
<tr>
<td>Europe/Kiev</td>
</tr>
<tr>
<td>Europe/Lisbon Portugal</td>
</tr>
<tr>
<td>Europe/London GB GB-Eire</td>
</tr>
<tr>
<td>Europe/Luxembourg</td>
</tr>
<tr>
<td>Europe/Madrid</td>
</tr>
<tr>
<td>Europe/Malta</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Europe/Minsk</td>
</tr>
<tr>
<td>Arctic/Longyearbyen</td>
</tr>
<tr>
<td>Atlantic/Jan_Mayen</td>
</tr>
<tr>
<td>Europe/Paris</td>
</tr>
<tr>
<td>Europe/Prague Europe/Bratislava</td>
</tr>
<tr>
<td>Europe/Riga</td>
</tr>
<tr>
<td>Europe/Rome Europe/San_Marino Europe/Vatican</td>
</tr>
<tr>
<td>Europe/Samara</td>
</tr>
<tr>
<td>Europe/Simferopol</td>
</tr>
<tr>
<td>Europe/Sofia</td>
</tr>
<tr>
<td>Europe/Stockholm</td>
</tr>
<tr>
<td>Europe/Tallinn</td>
</tr>
<tr>
<td>Europe/Tirane</td>
</tr>
<tr>
<td>Timezone</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Europe/Uzhgorod</td>
</tr>
<tr>
<td>Europe/Vaduz</td>
</tr>
<tr>
<td>Europe/Vienna</td>
</tr>
<tr>
<td>Europe/Vilnius</td>
</tr>
<tr>
<td>Europe/Warsaw</td>
</tr>
<tr>
<td>Europe/Zaporozhye</td>
</tr>
<tr>
<td>Europe/Zurich</td>
</tr>
</tbody>
</table>

**Indian**

<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indian/Antananarivo</td>
</tr>
<tr>
<td>Indian/Chagos</td>
</tr>
<tr>
<td>Indian/Christmas</td>
</tr>
<tr>
<td>Indian/Cocos</td>
</tr>
<tr>
<td>Indian/Comoro</td>
</tr>
<tr>
<td>Indian/Kerguelen</td>
</tr>
<tr>
<td>Indian/Mahe</td>
</tr>
<tr>
<td>Indian/Maldives</td>
</tr>
<tr>
<td>Indian/Mauritius</td>
</tr>
<tr>
<td>Indian/Mayotte</td>
</tr>
<tr>
<td>Indian/Reunion</td>
</tr>
</tbody>
</table>
### Pacific

<table>
<thead>
<tr>
<th>Time Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pacific/Apiia</td>
</tr>
<tr>
<td>Pacific/Auckland NZ</td>
</tr>
<tr>
<td>Pacific/Chatham NZ-CHAT</td>
</tr>
<tr>
<td>Pacific/Easter Chile/EasterIsland</td>
</tr>
<tr>
<td>Pacific/Efate</td>
</tr>
<tr>
<td>Pacific/Enderbury</td>
</tr>
<tr>
<td>Pacific/Fakaofo</td>
</tr>
<tr>
<td>Pacific/Fiji</td>
</tr>
<tr>
<td>Pacific/Funafuti</td>
</tr>
<tr>
<td>Pacific/Galapagos</td>
</tr>
<tr>
<td>Pacific/Gambier SystemV/YST9</td>
</tr>
<tr>
<td>Pacific/Guadalcanal</td>
</tr>
<tr>
<td>Pacific/Guam</td>
</tr>
<tr>
<td>Pacific/Honolulu HST SystemV/HST10 US/Hawaii</td>
</tr>
<tr>
<td>Pacific/Johnston</td>
</tr>
<tr>
<td>Pacific/Kirimiti</td>
</tr>
<tr>
<td>Pacific/Kosrae</td>
</tr>
<tr>
<td>Pacific/Kwajalein Kwajalein</td>
</tr>
<tr>
<td>Pacific/Majuro</td>
</tr>
<tr>
<td>Pacific/Marquesas</td>
</tr>
<tr>
<td>Timezone</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Pacific/Midway</td>
</tr>
<tr>
<td>Pacific/Nauru</td>
</tr>
<tr>
<td>Pacific/Niue</td>
</tr>
<tr>
<td>Pacific/Norfolk</td>
</tr>
<tr>
<td>Pacific/Noumea</td>
</tr>
<tr>
<td>Pacific/Pago_Pago</td>
</tr>
<tr>
<td>Pacific/Samoa US/Samoa</td>
</tr>
<tr>
<td>Pacific/Palau</td>
</tr>
<tr>
<td>Pacific/Pitcairn SystemV/PST8</td>
</tr>
<tr>
<td>Pacific/Ponape</td>
</tr>
<tr>
<td>Pacific/Port_Moresby</td>
</tr>
<tr>
<td>Pacific/Rarotonga</td>
</tr>
<tr>
<td>Pacific/Saipan</td>
</tr>
<tr>
<td>Pacific/Tahiti</td>
</tr>
<tr>
<td>Pacific/Tarawa</td>
</tr>
<tr>
<td>Pacific/Tongatapu</td>
</tr>
<tr>
<td>Pacific/Truk</td>
</tr>
<tr>
<td>Pacific/Wake</td>
</tr>
<tr>
<td>Pacific/Wallis</td>
</tr>
<tr>
<td>Pacific/Yap</td>
</tr>
</tbody>
</table>
Welcome to Getting Started. This guide walks you through the process of configuring a Vertica Analytics Platform database and running typical queries.

Important: Before you start, you should be familiar with Vertica concepts.
Using This Guide

Getting Started shows how to set up a Vertica database and run simple queries that perform common database tasks.

Who Should Use This Guide?

Getting Started targets anyone who wants to learn how to create and run a Vertica database. This guide requires no special knowledge at this point, although a rudimentary knowledge of basic SQL commands is useful when you begin to run queries.

What You Need

The examples in this guide require one of the following:

- Vertica installed on one host or a cluster of hosts. Vertica recommends a minimum of three hosts in the cluster.
- Vertica installed on a virtual machine (VM).

For further instructions about installation, see Installing Vertica.

Accessing Your Database

You access your database with an SSH client or the terminal utility in your Linux console, such as vsql. Throughout this guide, you use the following user interfaces:

- Linux command line (shell) interface
- Vertica Administration Tools
- vsql client interface
- Vertica Management Console
Downloading and Starting the Vertica Community Edition Virtual Machine

For a hands-on introduction to Vertica, you can use the Vertica Community Edition Virtual Machine (Vertica CE VM), which is available for download on my.vertica.com.

The Vertica CE VM is a preconfigured Linux environment that includes:

- Vertica Community Edition with the Vmart example database
- Management Console
- Administration Tools and vsql
- A user guide that guides you through a series of common tasks

After you download and install the VM and the Vertica documentation, the environment runs locally on your computer and does not require internet access.

Note: The Vertica CE VM is not supported for production use.

Requirements

- A virtualization application: either VMware or Oracle VM VirtualBox.
- Between 20 and 50 GB of available disk space on the host computer. The VM can use up to a maximum of 50 GB, depending on the operations you perform.

The Vertica CE VM is available in two virtualization formats: one for VMware, the other for Oracle VM VirtualBox.

- VMware: vertica_community_edition-9.0.1_vmx.zip uses 10GB of disk space after you download and unzip it on your computer. It uses an additional 10GB after you load it into VMware. You can reclaim half of that 20GB by deleting the unzipped directory after loading it into VMware.

- VirtualBox: vertica_community_edition-9.0.1.ova is read directly by VirtualBox and uses a total of 10GB of disk space.
The Vertica CE VM Environment

The Vertica CE VM is a CentOS 7.4 virtual machine. The VM is configured with 2 CPUs, 4 GB of RAM, and 50 GB HDD.

The Vertica database that is running in the VM has a single node.

Download the Vertica CE VM and the Vertica Documentation

1. Sign in to my.vertica.com or create an account.

2. Navigate to the Downloads page.

3. Download the Vertica CE VM for your virtualization platform:
   - Click VMware VMX File Format to download the VM for VMware.
   - Click Open Virtualization Format to download the VM for VirtualBox.

4. (Optional) Download the Vertica documentation:
   
   **Note:** The documentation ZIP file contains the entire Vertica documentation set. The documentation links in the user guide only work when the Vertica documentation is installed locally in the VM.

   - VMware: Download the documentation ZIP file:
     i. Open this link: https://my.vertica.com/documentation/vertica/9-0-x/.
     
     ii. Click Complete Documentation: Zipped HTML.
     
     iii. Click Save to save the ZIP file on your computer.

     The instructions in Install the Vertica Documentation on VMware explain how to copy/paste or drag the documentation ZIP file into the VM.

   - VirtualBox: You will download the documentation later as explained in Install the Vertica Documentation on VirtualBox.
Start Up and Log In to the Vertica CE VM

1. Locate the VM file that you downloaded.
   The file for VMware has a ZIP extension. The file for VirtualBox has an OVA extension or a
   TAR extension, depending on the browser you used for the download.
   - VMWare: Unzip the file.
   - VirtualBox: If the file has a TAR extension, change it to OVA.

2. Open the file in your VM player.
   - VMWare: Open the VMX file.
   - VirtualBox: Import the OVA file.

   Note: On initial startup only, VMware displays this message. Click I copied it to dismiss the message.

   ![Message](message.png)

   - VirtualBox: Import the OVA file.

3. Start up the VM.
   Note: Startup may take some time.

4. Log in to the VM with these credentials:
   User: Vertica DBA
   Password: password
Install the Vertica Documentation on VMware

1. On the VM desktop, double click home, then double click Vertica_Docs to navigate to /home/Vertica_Docs/.

   ![Home and Vertica_Docs Folders]

   **Important:** You must be logged in as dbadmin.

2. Navigate to the location where you downloaded the Vertica documentation ZIP file on your computer. The name of the ZIP file is:

   Vertica-user-doc_9.0.x-for-download.zip

3. To copy the Vertica documentation ZIP file to /home/Vertica_Docs/ in the VM, drag it or copy and paste it.

4. Open a terminal window in the VM.

5. Execute the following commands:

   ```
   $ cd /home/dbadmin/Vertica_Docs/
   $ unzip Vertica-user-doc_9.0.x-for-download.zip -d /home/dbadmin/Vertica_Docs/
   ```
Install the Vertica Documentation on VirtualBox

1. In the VM, open this link: https://my.vertica.com/documentation/vertica/9-0-x/.
   
   **Important:** You must be logged in as dbadmin.

2. Click Complete Documentation: Zipped HTML.

3. Click Save to save the ZIP file to the Downloads folder in the VM.

4. Move the ZIP file to this folder:

   /home/dbadmin/Vertica_Docs/

5. Right-click the ZIP file and select Extract here.
Using the Vertica Interfaces

Vertica provides a set of tools that allows you to perform administrative tasks quickly and easily. The administration tasks in Vertica can be done using the Management Console (MC) or the Administration Tools. The MC provides a unified view of your Vertica cluster through a browser connection, while the Administration Tools are implemented using Dialog, a graphical user interface that works in terminal (character-cell) windows.

Using Management Console

The Management Console provides some, but not all, of the functionality that the Administration Tools provides. In addition, the MC provides extended functionality not available in the Administration Tools, such as a graphical view of your Vertica database and detailed monitoring charts and graphs.

Most of the information you need to use MC is available on the MC interface, as seen in the following two screenshots. For installation instructions, see Installing and Configuring Management Console in the Installation Guide. For an introduction to MC functionality, architecture, and security, see Management Console in Vertica Concepts.
Running the Administration Tools

A man page is available for convenient access to Administration Tools details. If you are running as the dbadmin user, type `man admintools`. If you are running as a different user, type `man -M /opt/vertica/man admintools`. If possible, always run the Administration Tools using the database administrator account (dbadmin) on the administration host.

The Administration Tools interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. For a quick reference to keystrokes, see Using Keystrokes in the Administration Tools Interface in this guide.

When you run Administration Tools, the Main Menu dialog box appears with a dark blue background and a title on top. The screen captures used in this documentation set are cropped down to the dialog box itself, as shown in the following screenshot.
First Time Only

The first time you log in as the database administrator and run the Administration Tools, complete the following steps.

1. In the EULA (end-user license agreement) window, type accept to proceed. A window displays, requesting the location of the license key file you downloaded from the OpenText Web site. The default path is /tmp/vlicense.dat.

2. Type the absolute path to your license key (for example, /tmp/vlicense.dat) and click OK.

3. To return to the command line, select Exit and click OK.

Using Keystrokes in the Administration Tools Interface

The following table is a quick reference to keystroke usage in the Administration Tools interface. See Using the Administration Tools in the Administrator’s Guide for full details.

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Run selected command.</td>
</tr>
<tr>
<td>Tab</td>
<td>Cycle between OK, Cancel, Help, and menu.</td>
</tr>
<tr>
<td>Up/Down Arrow</td>
<td>Move cursor up and down in menu, window, or help file.</td>
</tr>
<tr>
<td>Space</td>
<td>Select item in list.</td>
</tr>
<tr>
<td>Character</td>
<td>Select corresponding command from menu.</td>
</tr>
</tbody>
</table>
Introducing the VMart Example Database

Vertica ships with a sample multi-schema database called the VMart Example Database, which represents a database that might be used by a large supermarket (VMart) to access information about its products, customers, employees, and online and physical stores. Using this example, you can create, run, optimize, and test a multi-schema database.

The VMart database contains the following schema:

- public (automatically created in any newly created Vertica database)
- store
- online_Sales

VMart Database Location and Scripts

If you installed Vertica from the RPM package, the VMart schema is installed in the /opt/vertica/examples/VMart_Schema directory. This folder contains the following script files that you can use to get started quickly. Use the scripts as templates for your own applications.

<table>
<thead>
<tr>
<th>Script/file name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmart_count_data.sql</td>
<td>SQL script that counts rows of all example database tables, which you can use to verify load.</td>
</tr>
<tr>
<td>vmart_define_schema.sql</td>
<td>SQL script that defines the logical schema for each table and referential integrity constraints.</td>
</tr>
<tr>
<td>vmart_gen.cpp</td>
<td>Data generator source code (C++).</td>
</tr>
<tr>
<td>vmart_gen</td>
<td>Data generator executable</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>vmart_load_data.sql</code></td>
<td>SQL script that loads the generated sample data to the corresponding tables using COPY DIRECT.</td>
</tr>
<tr>
<td><code>vmart_queries.sql</code></td>
<td>SQL script that contains concatenated sample queries for use as a training set for the Database Designer.</td>
</tr>
<tr>
<td><code>vmart_query_##.sql</code></td>
<td>SQL scripts that contain individual queries; for example, <code>vmart_query_01</code> through <code>vmart_query_09.sql</code></td>
</tr>
<tr>
<td><code>vmart_schema_drop.sql</code></td>
<td>SQL script that drops all example database tables.</td>
</tr>
</tbody>
</table>

For more information about the schema, tables, and queries included with the VMart example database, see the Appendix.
Installing and Connecting to the VMart Example Database

Follow the steps in this section to create the fully functioning, multi-schema VMart example database that you’ll use to run sample queries. The number of example databases you create within a single Vertica installation is limited only by the disk space available on your system; however, Vertica strongly recommends that you start only one example database at a time to avoid unpredictable results.

Vertica provides two options to install the example database:

- A quick installation that lets you create the example database and start using it immediately. See Quick Installation Using a Script in this guide for details. Use this method to bypass the schema and table creation processes and start querying immediately.

- An advanced-but-simple example database installation using the Administration Tools interface. See Advanced Installation in this guide for details. Use this method to better understand the database creation process and practice creating schema and tables, and loading data.

Note: Both installation methods create a database named VMart. If you try both installation methods, you will either need to drop the VMart database you created (see Restoring the Status of Your Host in this guide) or create the subsequent database with a new name. However, Vertica strongly recommends that you start only one example database at a time to avoid unpredictable results.

This tutorial uses Vertica-provided queries, but you can follow the same set of procedures later, when you create your own design and use your own queries file.

After you install the VMart database, the database has started. Connect to it using the steps in Step 3: Connecting to the Database.

Quick Installation Using a Script

The script you need to perform a quick installation is located in /opt/vertica/sbin and is called install_example. This script creates a database on the default port (5433), generates data, creates the schema and a default superprojection, and loads the data. The folder also contains a delete_example script, which stops and drops the database.
1. In a terminal window, log in as the database administrator.

   $ su dbadmin
   Password: (your password)

2. Change to the /examples directory.

   $ cd /opt/vertica/examples

3. Run the install script:

   $ /opt/vertica/sbin/install_example VMart

After installation, you should see the following:

```
[dbadmin@localhost examples]$ /opt/vertica/sbin/install_example VMart
Installing VMart example example database
Mon Jul 22 06:57:40 PDT 2013
Creating Database
Completed
Generating Data. This may take a few minutes.
Completed
Creating schema
Completed
Loading 5 million rows of data. Please stand by.
Completed
Removing generated data files
Example data
```

The example database log files, ExampleInstall.txt and ExampleDelete.txt, are written to /opt/vertica/examples/log.

To start using your database, continue to Connecting to the Database in this guide. To drop the example database, see Restoring the Status of Your Host in this guide.

Advanced Installation

To perform an advanced-but-simple installation, set up the VMart example database environment and then create the database using the Administration Tools or Management Console.

Note: If you installed the VMart database using the quick installation method, you cannot complete the following steps because the database has already been created.

To try the advanced installation, drop the example database (see Restoring the Status of Your Host on this guide) and perform the advanced Installation, or create a new example
database with a different name. However, Vertica strongly recommends that you install only one example database at a time to avoid unpredictable results.

The advanced installation requires the following steps:

- **Step 1: Setting Up the Example Environment**
- **Step 2: Creating the Example Database**
- **Step 3: Connecting to the Database**
- **Step 4: Defining the Database Schema**
- **Step 5: Loading Data**

**Step 1: Setting Up the Example Environment**

1. Stop all databases running on the same host on which you plan to install your example database.
   
   If you are unsure if other databases are running, run the Administration Tools and select View Cluster State. The State column should show DOWN values on pre-existing databases.
   
   If databases are running, click Stop Database in the Main Menu of the Administration Tools interface and click OK.

2. In a terminal window, log in as the database administrator:

   ```
   $ su dbadmin
   Password:
   ```

3. Change to the `/VMart_Schema` directory.

   ```
   $ cd /opt/vertica/examples/VMart_Schema
   ```
   
   Do not change directories while following this tutorial. Some steps depend on being in a specific directory.

4. Run the sample data generator.
5. Let the program run with the default parameters, which you can review in the README file.

```
$ ./vmart_gen

Using default parameters
datadirectory = ./
numfiles = 1
seed = 2
null = ''
timefile = Time.txt
numfactsalesrows = 5000000
numfactorderrows = 300000
numprodkeys = 60000
numstorekeys = 250
numpromomokeys = 1000
numvendkeys = 50
numcustkeys = 50000
numempkeys = 10000
numwarehousekeys = 100
numshippingkeys = 100
numonlinenepagekeys = 1000
numcallcenterkeys = 200
numfactonlinesalesrows = 5000000
numinventoryfactrows = 300000
gen_load_script = false
Data Generated successfully!
```

6. If the `vmart_gen` executable does not work correctly, recompile it as follows, and run the sample data generator script again.

```
Using default parameters
datadirectory = ./
numfiles = 1
seed = 2
null = ''
timefile = Time.txt
numfactsalesrows = 5000000
numfactorderrows = 300000
numprodkeys = 60000
numstorekeys = 250
numpromomokeys = 1000
numvendkeys = 50
numcustkeys = 50000
numempkeys = 10000
numwarehousekeys = 100
numshippingkeys = 100
numonlinenepagekeys = 1000
numcallcenterkeys = 200
numfactonlinesalesrows = 5000000
numinventoryfactrows = 300000
gen_load_script = false
Data Generated successfully!
```
Step 2: Creating the Example Database

To create the example database: use the Administration Tools or Management Console, as described in this section.

Creating the Example Database Using the Administration Tools

In this procedure, you create the example database using the Administration Tools. To use the Management Console, go to the next section.

Note: If you have not used Administration Tools before, see Running the Administration Tools in this guide.

1. Run the Administration Tools.

   $ /opt/vertica/bin/admintools

   or simply type admintools

2. From the Administration Tools Main Menu, click Configuration Menu and click OK.

3. Click Create Database and click OK.

4. Name the database VMart and click OK.

5. Click OK to bypass the password and click Yes to confirm.
There is no need for a database administrator password in this tutorial. When you create a production database, however, always specify an administrator password. Otherwise, the database is permanently set to trust authentication (no passwords).

6. Select the hosts you want to include from your Vertica cluster and click OK.

This example creates the database on a one-host cluster. Vertica recommends a minimum of three hosts in the cluster. If you are using the Vertica Community Edition, you are limited to three nodes.

![Select hosts for the database](image)

7. Click OK to select the default paths for the data and catalog directories.

![Database data directories](image)

- Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

- When you create a production database, you’ll likely specify other locations than the default. See Prepare Disk Storage Locations in the Administrator’s Guide for more information.

8. Since this tutorial uses a one-host cluster, a K-safety warning appears. Click OK.

![Database with 1 or 2 hosts cannot be k-safe and it may lose data if it crashes](image)

9. Click Yes to create the database.
During database creation, Vertica automatically creates a set of node definitions based on the database name and the names of the hosts you selected and returns a success message.

10. Click OK to close the Database VMart created successfully message.

Creating the Example Database Using Management Console

In this procedure, you create the example database using Management Console. To use the Administration Tools, follow the steps in the preceding section.

**Note:** To use Management Console, the console should already be installed and you should be familiar with its concepts and layout. See Using Management Console in this guide for a brief overview, or for detailed information, see Management Console in Vertica Concepts and Installing and Configuring Management Console in Installing Vertica.

1. Connect to Management Console and log in.

2. On the Home page, under Manage Information, click Existing Infrastructure to go to the Databases and Clusters page.
3. Click to select the appropriate existing cluster and click Create Database.
4. Follow the on-screen wizard, which prompts you to provide the following information:
   - Database name, which must be between 3–25 characters, starting with a letter, and followed by any combination of letters, numbers, or underscores.
   - (Optional) database administrator password for the database you want to create and connect to.
   - IP address of a node in your database cluster, typically the IP address of the administration host.

5. Click Next.

**Step 3: Connecting to the Database**

Regardless of the installation method you used, follow these steps to connect to the database.

1. As dbadmin, run the Administration Tools.

   $ /opt/vertica/bin/admintools

   or simply type admintools.

2. If you are already in the Administration Tools, navigate to the Main Menu page.

3. Select Connect to Database, click OK.

![Main Menu](image)

To configure and load data into the VMart database, complete the following steps:

- **Step 4: Defining the Database Schema**

- **Step 5: Loading Data**

If you installed the VMart database using the Quick Installation method, the schema, tables, and data are already defined. You can choose to drop the example database (see
Restoring the Status of Your Host in this guide) and perform the Advanced Installation, or continue straight to Querying Your Data in this guide.

Step 4: Defining the Database Schema

The VMart database installs with sample scripts with SQL commands that are intended to represent queries that might be used in a real business. The vmart_define_schema.sql script runs a script that defines the VMart schema and creates tables. You must run this script before you load data into the VMart database.

This script performs the following tasks:

- Defines two schemas in the VMart database schema: online_sales and store.
- Defines tables in both schemas.
- Defines constraints on those tables.

```
Vmart=> \i vmart_define_schema.sql
CREATE SCHEMA
CREATE SCHEMA
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
```

Step 5: Loading Data

Now that you have created the schemas and tables, you can load data into a table by running the vmart_load_data.sql script. This script loads data from the 15.tbl text files in opt/vertica/examples/VMart_Schema into the tables that vmart_design_schema.sql created.
It might take several minutes to load the data on a typical hardware cluster. Check the load status by monitoring the `vertica.log` file, as described in Monitoring Log Files in the Administrator's Guide.

```
VMart=> \i vmart_load_data.sql
Rows Loaded
---------
1826
(1 row)
Rows Loaded
---------
60000
(1 row)
Rows Loaded
---------
250
(1 row)
Rows Loaded
---------
1000
(1 row)
Rows Loaded
---------
50
(1 row)
Rows Loaded
---------
50000
(1 row)
Rows Loaded
---------
10000
(1 row)
Rows Loaded
---------
100
(1 row)
Rows Loaded
---------
100
(1 row)
Rows Loaded
---------
1000
(1 row)
Rows Loaded
---------
200
(1 row)
Rows Loaded
---------
```
<table>
<thead>
<tr>
<th>Rows Loaded</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5000000</td>
<td>(1 row)</td>
</tr>
<tr>
<td>300000</td>
<td>(1 row)</td>
</tr>
</tbody>
</table>

VMart=>
Querying Data

The VMart database installs with sample scripts that contain SQL commands that represent queries that might be used in a real business. Use basic SQL commands to query the database, or try out the following command. Once you’re comfortable running the example queries, you might want to write your own.

Note: The data that your queries return might differ from the example output shown in this guide because the sample data generator is random.

Type the following SQL command to return the values for five products with the lowest fat content in the Dairy department. The command selects the fat content from Dairy department products in the product_dimention table in the public schema, orders them from low to high and limits the output to the first five (the five lowest fat contents).

VMart => SELECT fat_content
FROM (SELECT DISTINCT fat_content
FROM product_dimension
WHERE department_description
IN ('Dairy') ) AS food
ORDER BY fat_content
LIMIT 5;

Your results will be similar to the following:

<table>
<thead>
<tr>
<th>fat_content</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>83</td>
</tr>
<tr>
<td>84</td>
</tr>
</tbody>
</table>

(5 rows)

The preceding example is from the vmart_query_01.sql file. You can execute more sample queries using the scripts that installed with the VMart database or write your own. For a list of the sample queries supplied with Vertica, see Appendix: VMart Example Database Schema, Tables, and Scripts.
Backing Up and Restoring the Database

Vertica supplies a comprehensive utility, the vbr Python script, that lets you back up and restore a full database, as well as create backups of specific schema or tables. The vbr utility creates backup directories during its initial execution; subsequently running the utility creates subdirectories.

The following information is intended to introduce the backup and restore functions. For more detailed information, see Backing Up and Restoring the Database in the Administrator’s Guide.

Backing Up the Database

Use vbr to save your data to a variety of locations:

- A local directory on the nodes in a cluster
- One or more hosts outside of the cluster
- A different Vertica cluster (effectively cloning your database)

Note: Creating a database backup on a different cluster does not provide disaster recovery. The cloned database you create with vbr is entirely separate from the original, and is not kept synchronized with the database from which it is cloned.

When to Back up the Database

In addition to any guidelines established by your organization, Vertica recommends that you back up your database:

- Before you upgrade Vertica to another release.
- Before you drop a partition.
- After you load a large volume of data.
- If the epoch in the latest backup is earlier than the current ancient history mark (AHM).
Before and after you add, remove, or replace nodes in your database cluster.

After recovering a cluster from a crash.

**Note:** When you restore a database backup, you must restore to a cluster that is identical to the one where you created the backup. For this reason, always create a new backup after adding, removing, or replacing nodes.

Ideally, create regular backups of your full database. You can run the Vertica `vbr` utility from a cron job or other task scheduler.

## Creating the Backup Configuration File

The `vbr` utility uses a configuration file for the information required to back up and restore a full- or object-level backup. The configuration file defines where the database backup is saved, the temporary directories it uses, and which nodes, schema, and/or tables in the database are to be backed up. You cannot run `vbr` without a configuration file, and no default file exists.

To invoke the script to set up a configuration file, enter this command:

```
$ vbr --setupconfig
```

The script prompts you to answer the following questions regarding the configuration file. Type `Enter` to accept the default value in parentheses. See [VBR Configuration File Reference](#) in the Administrator’s Guide for information about specific questions.

```
[dbadmin@localhost ~]$ /opt/vertica/bin/vbr --setupconfig
Snapshot name (backup_snapshot): fullbak1
Number of restore points (1): 3
Specify objects (no default):
Object restore mode (coexist, createOrReplace or create) (createOrReplace):
Vertica user name (dbadmin):
Save password to avoid runtime prompt? (n) [y/n]: y
Password to save in vbr config file (no default):
Node v_vmart_node0001
Backup host name (no default): 194.66.82.11
Backup directory (no default): /home/dbadmin/backups
Config file name (fullbak1.ini):
Password file name (no default value) (no default):pwdfile
Change advanced settings? (n) [y/n]: n
```
Saved vbr configuration to fullbak1.ini.  
Saved vbr database password to pwdfile.ini.

After you answer the required questions, vbr generates a configuration file with the information you supplied. Use the Config file name you specified when you run the --task backup or other commands. The vbr utility uses the configuration file contents for both backup and restore tasks.

Creating Full and Incremental Backups

Before you create a database backup, ensure the following:

- Your database is running.
- All of the backup hosts are up and available.
- The backup location host has sufficient disk space to store the backups.
- The user who starts the utility has write access to the target directories on the host backup location.

Run the vbr script from a terminal using the database administrator account from an initiator node in your database cluster. You cannot run the utility as root.

Use the --task backup and --config-file filename directives as shown in this example.

```
[release@qco55srv01:/scratch_b/qa/vertica/QA/VT_Scenario 0]$
vbr -t backup --config $FULLBAK_CONFIG
Starting backup of database VTDB.
Participating nodes: v_vmart_node0001, v_vmart_node0002, v_vmart_node0003, v_vmart_node0004.
Snapshotting database.
Snapshot complete.
Approximate bytes to copy: 2315056043 of 2356089422 total.
[==================================================] 100%
Copying backup metadata.
Finalizing backup.
Backup complete!
```

By default, there is no screen output other than the progress bar.

If you do not specify a configuration file, the vbr utility searches for one at this location:

```
/opt/vertica/config/vbr.ini
```

If the utility does not locate the configuration you specify, it searches for one at opt/vertica/config/vbr.ini. If no file exists there, it fails with an error.
The first time you run the vbr utility, it performs a full backup; subsequent runs with the same configuration file create an incremental backup. When creating incremental backups, the utility copies new storage containers, which can include data that existed the last time you backed up the database, along with new and changed data since then. By default, vbr saves one archive backup, unless you set the restorePointLimit parameter value in the configuration file to a value greater than 1.

**Restoring the Database**

To restore a full database backup, ensure that:

- The database is down.
- All of the backup hosts are up and available.
- The backup directory exists and contains the backups from which to restore.

To begin a full database backup restore, log in using the database administrator’s account. You cannot run the utility as root. For detailed instructions on restoring a database, refer to [Recovering the Database](#).
Using Database Designer to Create a Comprehensive Design

The Vertica Database Designer:

- Analyzes your logical schema, sample data, and, optionally, your sample queries.
- Creates a physical schema design (a set of projections) that can be deployed automatically or manually.
- Can be used by anyone without specialized database knowledge.
- Can be run and rerun any time for additional optimization without stopping the database.
- Uses strategies to provide optimal query performance and data compression.

Use Database Designer to create a comprehensive design, which allows you to create new projections for all tables in your database.

You can also use Database Designer to create an incremental design, which creates projections for all tables referenced in the queries you supply. For more information, see Incremental Design in the Administrator’s Guide.

You can create a comprehensive design with Database Designer using Management Console or through Administration Tools. You can also choose to run Database Designer programmatically (See About Running Database Designer Programmatically).

This section shows you how to:

- Running Database Designer with Management Console
- Run Database Designer with Administration Tools

Running Database Designer with Management Console

In this tutorial, you'll create a comprehensive design with Database Designer through the Management Console interface. If, in the future, you have a query that you want to optimize, you can create an enhanced (incremental) design with additional projections. You can tune
these projections specifically for the query you provide. See Comprehensive Design in the Administrator's Guide for more information.

Note: To run Database Designer outside Administration Tools, you must be a dbadmin user. If you are not a dbadmin user, you must have the DBDUSER role assigned to you and own the tables for which you are designing projections.

You can choose to create the design manually or use the wizard. To create a design manually, see Creating a Design Manually in the Administrator's Guide.

Set your browser so that it does not cache pages. If a browser caches pages, you may not be able to see the new design added.

Follow these steps to use the wizard to create the comprehensive design in Management Console:

1. Log in to Management Console.
2. Verify that your database is up and running.
3. Choose the database for which you want to create the design. You can find the database under the Recent Databases section or by clicking Existing Infrastructure to reach the Databases and Clusters page.

The database overview page opens:

![Database Overview Page]

4. At the bottom of the screen, click the Design button.
5. In the New Design dialog box, enter the design name.

6. Click Wizard to continue.

7. Create an initial design. For Design Type, select Comprehensive and click Next.

8. In the Optimization Objective window, select Balance Load and Performance to create a design that is balanced between database size and query performance. Click Next.

9. Select the schemas. Because the VMart design is a multi-schema database, select all three schemas (public, store, and online_sales) for your design in the Select Sample Data window. Click Next.
If you include a schema that contains tables without data, the design could be suboptimal. You can choose to continue, but Vertica recommends that you deselect the schemas that contain empty tables before you proceed.

10. Choose the K-safety value for your design. The K-Safety value determines the number of buddy projections you want database designer to create.

11. Choose Analyze Correlations Mode. Analyze Correlations Mode determines if Database Designer analyzes and considers column correlations when creating the design.

   - Ignore: When creating a design, ignore any column correlations in the specified tables.
   - Consider existing: Consider the existing correlations in the tables when creating the design. If you set the mode to 1, and there are no existing correlations, Database Designer does not consider correlations.
   - Analyze missing: Analyze column correlations on tables where the correlation analysis was not previously performed. When creating the design, consider all column correlations (new and existing).
   - Analyze all: Analyze all column correlations in the tables and consider them when creating the design. Even if correlations exist for a table, reanalyze the table for correlations.

   Click Next.

12. Submit query files to Database Designer in one of two ways:
- Supply your own query files by selecting the Browse button.

- Click Use Query Repository, which submits recently executed queries from the QUERY_REQUESTS system table.

Click Next.

13. In the Execution Options window, select all the options you want. You can select all three options or fewer.

The three options are:

- Analyze statistics: Select this option to run statistics automatically after design deploy to help Database Designer make more optimal decisions for its proposed design.

- Auto-build: Select this option to run Database Designer as soon as you complete the wizard. This option only builds the proposed design.

- Auto-deploy: Select this option for auto-build designs that you want to deploy automatically.

14. Click Submit Design.

The Database Designer page opens:

- If you chose to automatically deploy your design, Database Designer executes in the background.

- If you did not select the Auto-build or Auto-deploy options, you can click Build Design or Deploy Design on the Database Designer page.
15. In the My Designs pane, view the status of your design:

- When the deployment completes, the My Design pane shows Design Deployed.
- The event history window shows the details of the design build and deployment.

To run Database Designer with Administration Tools, see Run Database Designer with Administration Tools in this guide.

**Run Database Designer with Administration Tools**

In this procedure, you create a comprehensive design with Database Designer using the Administration Tools interface. If, in the future, you have a query that you want to optimize, you can create an enhanced (incremental) design with additional projections. You can tune these projections specifically for the query you provide. See Incremental Design in the Administrator’s Guide for more information.

Follow these steps to create the comprehensive design using Database Designer in Administration Tools:

1. If you are not in Administration Tools, exit the vsq1 session and access Administration Tools:

   - Type \q to exit vsq1.
   - Type admintools to access the Administration Tools Main Menu.
2. Start the database for which you want to create a design.

3. From the Main Menu, click Configuration Menu and then click OK.

4. From the Configuration Menu, click Run Database Designer and then click OK.

5. When the Select a database for design dialog box opens, select VMart and then click OK.

   ![Select a database for design dialog box](image)

   If you are prompted to enter the password for the database, click OK to bypass the message. Because no password was assigned when you installed the VMart database, you do not need to enter one now.

6. Click OK to accept the default directory for storing Database Designer output and log files.

   ![Enter directory for Database Designer output](image)

7. In the Database Designer window, enter a name for the design, for example, `vmart_design`, and click OK. Design names can contain only alphanumeric characters or underscores. No other special characters are allowed.
8. Create a complete initial design. In the Design Type window, click Comprehensive and click OK.

9. Select the schemas. Because the VMart design is a multi-schema database, you can select all three schemas (online_sales, public, and store) for your design. Click OK.

If you include a schema that contains tables without data, the Administration Tools notifies you that designing for tables without data could be suboptimal. You can choose to continue, but Vertica recommends that you deselect the schemas that contain empty tables before you proceed.

10. In the Design Options window, accept all three options and click OK.
The three options are:

- **Optimize with queries**: Supplying the Database Designer with queries is especially important if you want to optimize the database design for query performance. Vertica recommends that you limit the design input to 100 queries.

- **Update statistics**: Accurate statistics help the Database Designer choose the best strategy for data compression. If you select this option, the database statistics are updated to maximize design quality.

- **Deploy design**: The new design deploys automatically. During deployment, new projections are added, some existing projections retained, and any necessary existing projections removed. Any new projections are refreshed to populate them with data.

11. **Because you selected the Optimize with queries option**, you must enter the full path to the file containing the queries that will be run on your database. In this example, it is:

```
/opt/vertica/examples/VMart_Schema/vmart_queries.sql
```

The queries in the query file must be delimited with a semicolon (;).

12. **Choose the K-safety value you want and click OK**. The design K-Safety determines the number of buddy projections you want database designer to create.
If you create a comprehensive design on a single node, you are not prompted to enter a K-safety value.

13. In the Optimization Objective window, select Balanced query/load performance to create a design that is balanced between database size and query performance. Click OK.

14. When the informational message displays, click Proceed.

Database Designer automatically performs these actions:

- Sets up the design session.
- Examines table data.
- Loads queries from the query file you provided (in this example, /opt/vertica/examples/VMart_Schema/vmart_queries.sql).
- Creates the design.

Deploys the design or saves a SQL file containing the commands to create the design, based on your selections in the Design Options window.

Depending on system resources, the design process could take several minutes. You should allow this process to complete uninterrupted. If you must cancel the session, use Ctrl+C.
15. When Database Designer finishes, press Enter to return to the Administration Tools menu. Examine the steps taken to create the design. The files are in the directory you specified to store the output and log files. In this example, that directory is /opt/vertica/examples/VMartSchema. For more information about the script files, see About Database Designer, in the Administrator's Guide.

For additional information about managing your designs, see Creating a Database Design in the Administrator's Guide.
Restoring the Status of Your Host

When you finish the tutorial, you can restore your host machines to their original state. Use the following instructions to clean up your host and start over from scratch.

Stopping and Dropping the Database

Follow these steps to stop and/or drop your database. A database must be stopped before it can be dropped.

1. If connected to the database, disconnect by typing \q.
2. In the Administration Tools Main Menu dialog box, click Stop Database and click OK.
3. In the Select database to stop window, select the database you want to stop and click OK.
4. After stopping the database, click Configuration Menu and click OK.
5. Click Drop Database and click OK.
6. In the Select database to drop window, select the database you want to drop and click OK.
7. Click Yes to confirm.
8. In the next window type yes (lowercase) to confirm and click OK.

Alternatively, use the delete_example script, which stops and drops the database:

1. If connected to the database, disconnect by typing \q.
2. In the Administration Tools Main Menu dialog box, select Exit.
3. Log in as the database administrator.
4. Change to the /examples directory.

```
$ cd /opt/vertica/examples
```
5. Run the delete_example script.

```
$ /opt/vertica/sbin/delete_example Vmart
```
Uninstalling Vertica

See Uninstalling Vertica.

Optional Steps

You can also choose to:

- Remove the dbadmin account on all cluster hosts.
- Remove any example database directories you created.
Changing the GUI Appearance

The appearance of the graphical user interface (GUI) depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on a Windows platform.

Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, verify that your window is at least 81 characters wide and 23 characters high.

If you are using PuTTY, take these steps to make the Administration Tools look like the screen captures in this document.

1. In a PuTTY window, right-click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Appearance.
4. In the Font settings, click the Change... button.
6. Click Apply.

Repeat these steps for each existing session that you use to run the Administration Tools.

You can also change the translation to support UTF-8.

1. In a PuTTY window, right-click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Translation.
4. In the Received data assumed to be in which character set drop-down menu, select UTF-8.
5. Click Apply.
Appendix: VMart Example Database Schema, Tables, and Scripts

The Appendix provides detailed information about the VMart example database’s schema, tables, and scripts.

The VMart example database contains three different schemas:

- public
- store
- online_sales

The term “schema” has several related meanings in Vertica:

- In SQL statements, a schema refers to named namespace for a logical schema.
- Logical schema refers to a set of tables and constraints.
- Physical schema refers to a set of projections.

Each schema contains tables that are created and loaded during database installation. See the schema maps for a list of tables and their contents:

- public Schema Map
- store Schema Map
- online_sales Schema Map

The VMart database installs with sample scripts that contain SQL commands that represent queries that might be used in a real business. The sample scripts are available in the Sample Scripts section of this Appendix. Once you’re comfortable running the example queries, you might want to write your own.

Tables

The three schemas in the VMart database include the following tables:

<table>
<thead>
<tr>
<th>public Schema</th>
<th>store Schema</th>
<th>online_sales</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Schema</th>
<th>Schema</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>inventory_fact</td>
<td>store_orders_fact</td>
<td>online_sales_fact</td>
</tr>
<tr>
<td>customer_dimension</td>
<td>store_sales_fact</td>
<td>call_center_dimension</td>
</tr>
<tr>
<td>date_dimension</td>
<td>store_dimension</td>
<td>online_page_dimension</td>
</tr>
<tr>
<td>employee_dimension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>product_dimension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>promotion_dimension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shipping_dimension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vendor_dimension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>warehouse_dimension</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**public Schema Map**

The public schema is a snowflake schema. The following graphic illustrates the public schema and its relationships with tables in the online_sales and store schemas.
inventory_fact

This table contains information about each product in inventory.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_version</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>warehouse_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>qty_in_stock</td>
<td>INTEGER</td>
<td>No</td>
</tr>
</tbody>
</table>
customer_dimension

This table contains information about all the retail chain’s customers.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>customer_type</td>
<td>VARCHAR(16)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_name</td>
<td>VARCHAR(256)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_gender</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
<tr>
<td>title</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
<tr>
<td>household_id</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_address</td>
<td>VARCHAR(256)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_city</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_state</td>
<td>CHAR(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_region</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>marital_status</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_age</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>number_of_children</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>annual_income</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>occupation</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>largest_bill_amount</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>store_membership_card</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>customer_since</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>deal_stage</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>deal_size</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
</tbody>
</table>
date_dimension

This table contains information about dates. It is generated from a file containing correct date/time data.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>full_date_description</td>
<td>VARCHAR(18)</td>
<td>Yes</td>
</tr>
<tr>
<td>day_of_week</td>
<td>VARCHAR(9)</td>
<td>Yes</td>
</tr>
<tr>
<td>day_number_in_calendar_month</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>day_number_in_calendar_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>day_number_in_fiscal_month</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>day_number_in_fiscal_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>last_day_in_week_indicator</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>last_day_in_month_indicator</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_week_number_in_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_month_name</td>
<td>VARCHAR(9)</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_month_number_in_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_year_month</td>
<td>CHAR(7)</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_quarter</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_year_quarter</td>
<td>CHAR(7)</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_half_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>calendar_year</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
</tbody>
</table>
This table contains information about all the people who work for the retail chain.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>employee_gender</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
<tr>
<td>courtesy_title</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_first_name</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_middle_initial</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_last_name</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_age</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>hire_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_street_address</td>
<td>VARCHAR(256)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_city</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_state</td>
<td>CHAR(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>employee_region</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>job_title</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>reports_to</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>salaried_flag</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>annual_salary</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>hourly_rate</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### product_dimension

This table describes all products sold by the department store chain.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>product_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_version</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_description</td>
<td>VARCHAR(128)</td>
<td>Yes</td>
</tr>
<tr>
<td>sku_number</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>category_description</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>department_description</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>package_type_description</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>package_size</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>fat_content</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>diet_type</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>weight</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>weight_units_of_measure</td>
<td>CHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>shelf_width</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>shelf_height</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>shelf_depth</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>product_price</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>product_cost</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>lowest_competitor_price</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>highest_competitor_price</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### average_competitor_price
- **Data Type:** INTEGER
- **NULLs:** Yes

### discontinued_flag
- **Data Type:** INTEGER
- **NULLs:** Yes

---

**promotion_dimension**

This table describes every promotion ever done by the retail chain.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>promotion_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>promotion_name</td>
<td>VARCHAR(128)</td>
<td>Yes</td>
</tr>
<tr>
<td>price_reduction_type</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>promotion_media_type</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>ad_type</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>display_type</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>coupon_type</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>ad_media_name</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>display_provider</td>
<td>VARCHAR(128)</td>
<td>Yes</td>
</tr>
<tr>
<td>promotion_cost</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>promotion_begin_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>promotion_end_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

**shipping_dimension**

This table contains information about shipping companies that the retail chain uses.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>shipping_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
</tbody>
</table>
### vendor_dimension

This table contains information about each vendor that provides products sold through the retail chain.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>vendor_name</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>vendor_address</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>vendor_city</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>vendor_state</td>
<td>CHAR(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>vendor_region</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>deal_size</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>last Deal Update</td>
<td>DATE</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### warehouse_dimension

This table provides information about each of the chain’s warehouses.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>warehouse_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>warehouse_name</td>
<td>VARCHAR(20)</td>
<td>Yes</td>
</tr>
<tr>
<td>warehouse_address</td>
<td>VARCHAR(256)</td>
<td>Yes</td>
</tr>
<tr>
<td>warehouse_city</td>
<td>VARCHAR(60)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The store schema is a snowflake schema that contains information about the retail chain’s bricks-and-mortar stores. The following graphic illustrates the store schema and its relationship with tables in the public schema.
store_orders_fact

This table contains information about all orders made at the company's brick-and-mortar stores.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>product_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_version</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>store_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>vendor_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>employee_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>order_number</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>date_ordered</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>date_shipped</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>expected_delivery_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>date_delivered</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>quantity_ordered</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>quantity_delivered</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>shipper_name</td>
<td>VARCHAR(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>unit_price</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>shipping_cost</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>total_order_cost</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>quantity_in_stock</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>reorder_level</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>overstock_ceiling</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
</tbody>
</table>
store_sales_fact

This table contains information about all sales made at the company’s brick-and-mortar stores.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_version</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>store_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>promotion_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>customer_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>employee_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>pos_transaction_number</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>sales_quantity</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>sales_dollar_amount</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>cost_dollar_amount</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>gross_profit_dollar_amount</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>transaction_type</td>
<td>VARCHAR(16)</td>
<td>Yes</td>
</tr>
<tr>
<td>transaction_time</td>
<td>TIME</td>
<td>Yes</td>
</tr>
<tr>
<td>tender_type</td>
<td>VARCHAR(8)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

store_dimension

This table contains information about each brick-and-mortar store within the retail chain.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
</tbody>
</table>
The `online_sales` schema is a snowflake schema that contains information about the retail chains. The following graphic illustrates the `online_sales` schema and its relationship with tables in the public schema.
online_sales_fact

This table describes all the items purchased through the online store front.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>sale_date_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>ship_date_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>product_version</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>customer_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>call_center_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>online_page_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>shipping_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>warehouse_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>promotion_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>NULLs</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>pos_transaction_number</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>sales_quantity</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>sales_dollar_amount</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
<tr>
<td>ship_dollar_amount</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
<tr>
<td>net_dollar_amount</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
<tr>
<td>cost_dollar_amount</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
<tr>
<td>gross_profit_dollar_amount</td>
<td>FLOAT</td>
<td>Yes</td>
</tr>
<tr>
<td>transaction_type</td>
<td>VARCHAR(16)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**call_center_dimension**

This table describes all the chain’s call centers.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_center_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>cc_closed_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_open_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_date</td>
<td>VARCHAR(50)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_class</td>
<td>VARCHAR(50)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_employees</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_hours</td>
<td>CHAR(20)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_manager</td>
<td>VARCHAR(40)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_address</td>
<td>VARCHAR(256)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_city</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_state</td>
<td>CHAR(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>cc_region</td>
<td>VARCHAR(64)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
online_page_dimension

This table describes all the pages in the online store front.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>NULLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>online_page_key</td>
<td>INTEGER</td>
<td>No</td>
</tr>
<tr>
<td>start_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>end_date</td>
<td>DATE</td>
<td>Yes</td>
</tr>
<tr>
<td>page_number</td>
<td>INTEGER</td>
<td>Yes</td>
</tr>
<tr>
<td>page_description</td>
<td>VARCHAR(100)</td>
<td>Yes</td>
</tr>
<tr>
<td>page_type</td>
<td>VARCHAR(100)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Sample Scripts

You can create your own queries, but the VMart example directory includes sample query script files to help you get started quickly.

You can find the following sample scripts at this path /opt/vertica/examples/VMart_Schema.

To run any of the scripts, enter

```
=> \i <script_name>
```

Alternatively, type the commands from the script file manually.

**Note:** The data that your queries return might differ from the example output shown in this guide because the sample data generator is random.

vmart_query_01.sql

```
-- vmart_query_01.sql
-- FROM clause subquery
```
-- Return the values for five products with the
-- lowest-fat content in the Dairy department

SELECT fat_content
FROM (  
    SELECT DISTINCT fat_content  
    FROM product_dimension  
    WHERE department_description  
        IN ('Dairy')  
    ) AS food  
ORDER BY fat_content  
LIMIT 5;

Output

<table>
<thead>
<tr>
<th>fat_content</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>83</td>
</tr>
<tr>
<td>84</td>
</tr>
</tbody>
</table>
(5 rows)

vsmart_query_02.sql

-- vsmart_query_02.sql
-- WHERE clause subquery
-- Asks for all orders placed by stores located in Massachusetts
-- and by vendors located elsewhere before March 1, 2003:

SELECT order_number, date_ordered
FROM store.store_orders_fact orders
WHERE orders.store_key IN (  
    SELECT store_key  
    FROM store.store_dimension  
    WHERE store_state = 'MA')  
AND orders.vendor_key NOT IN (  
    SELECT vendor_key  
    FROM public.vendor_dimension  
    WHERE vendor_state = 'MA')  
AND date_ordered < '2012-03-01';

Output

<table>
<thead>
<tr>
<th>order_number</th>
<th>date_ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>53019</td>
<td>2012-02-10</td>
</tr>
<tr>
<td>222168</td>
<td>2012-02-05</td>
</tr>
<tr>
<td>160881</td>
<td>2012-01-08</td>
</tr>
<tr>
<td>106922</td>
<td>2012-02-07</td>
</tr>
<tr>
<td>246465</td>
<td>2012-02-10</td>
</tr>
<tr>
<td>234218</td>
<td>2012-02-03</td>
</tr>
<tr>
<td>263119</td>
<td>2012-01-04</td>
</tr>
<tr>
<td>73015</td>
<td>2012-01-01</td>
</tr>
</tbody>
</table>
vmart_query_03.sql

-- vmart_query_03.sql
-- noncorrelated subquery
-- Requests female and male customers with the maximum
-- annual income from customers
SELECT customer_name, annual_income 
FROM public.customer_dimension
WHERE (customer_gender, annual_income) IN ( 
    SELECT customer_gender, MAX(annual_income) 
    FROM public.customer_dimension 
    GROUP BY customer_gender);

Output

customer_name     | annual_income
-------------------|----------------
James M. McNulty   | 999979
Emily G. Vogel    | 999998
(2 rows)

vmart_query_04.sql

-- vmart_query_04.sql
-- IN predicate
-- Find all products supplied by stores in MA
SELECT DISTINCT s.product_key, p.product_description 
FROM store.store_sales_fact s, public.product_dimension p 
WHERE s.product_key = p.product_key 
AND s.product_version = p.product_version AND s.store_key IN ( 
    SELECT store_key 
    FROM store.store_dimension)
WHERE store_state = 'MA')
ORDER BY s.product_key;

Output

<table>
<thead>
<tr>
<th>product_key</th>
<th>product_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Brand #1 butter</td>
</tr>
<tr>
<td>1</td>
<td>Brand #2 bagels</td>
</tr>
<tr>
<td>2</td>
<td>Brand #3 lamb</td>
</tr>
<tr>
<td>2</td>
<td>Brand #4 brandy</td>
</tr>
<tr>
<td>2</td>
<td>Brand #5 golf clubs</td>
</tr>
<tr>
<td>2</td>
<td>Brand #6 chicken noodle soup</td>
</tr>
<tr>
<td>3</td>
<td>Brand #10 ground beef</td>
</tr>
<tr>
<td>3</td>
<td>Brand #11 vanilla ice cream</td>
</tr>
<tr>
<td>3</td>
<td>Brand #7 canned chicken broth</td>
</tr>
<tr>
<td>3</td>
<td>Brand #8 halibut</td>
</tr>
<tr>
<td>3</td>
<td>Brand #9 camera case</td>
</tr>
<tr>
<td>4</td>
<td>Brand #12 rash ointment</td>
</tr>
<tr>
<td>4</td>
<td>Brand #13 low fat milk</td>
</tr>
<tr>
<td>4</td>
<td>Brand #14 chocolate chip cookies</td>
</tr>
<tr>
<td>4</td>
<td>Brand #15 silver polishing cream</td>
</tr>
<tr>
<td>5</td>
<td>Brand #16 cod</td>
</tr>
<tr>
<td>5</td>
<td>Brand #17 band aids</td>
</tr>
<tr>
<td>6</td>
<td>Brand #18 bananas</td>
</tr>
<tr>
<td>6</td>
<td>Brand #19 starch</td>
</tr>
<tr>
<td>6</td>
<td>Brand #20 vegetable soup</td>
</tr>
<tr>
<td>6</td>
<td>Brand #21 bourbon</td>
</tr>
</tbody>
</table>

vmart_query_05.sql

-- vmart_query_05.sql
-- EXISTS predicate
-- Get a list of all the orders placed by all stores on
-- January 2, 2003 for the vendors with records in the
-- vendor_dimension table
SELECT store_key, order_number, date_ordered
FROM store.store_orders_fact
WHERE EXISTS (SELECT 1
               FROM public.vendor_dimension
               WHERE public.vendor_dimension.vendor_key = store.store_orders_fact.vendor_key)
AND date_ordered = '2012-01-02';

Output

<table>
<thead>
<tr>
<th>store_key</th>
<th>order_number</th>
<th>date_ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>151837</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>123</td>
<td>238372</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>242</td>
<td>263973</td>
<td>2012-01-02</td>
</tr>
</tbody>
</table>
vmart_query_06.sql

-- vmart_query_06.sql
-- EXISTS predicate
-- Orders placed by the vendor who got the best deal
-- on January 4, 2004

SELECT store_key, order_number, date_ordered
FROM store.store_orders_fact ord, public.vendor_dimension vd
WHERE ord.vendor_key = vd.vendor_key
AND vd.deal_size IN (
    SELECT MAX(deal_size)
    FROM public.vendor_dimension)
AND date_ordered = '2013-01-04';

```
Output

<table>
<thead>
<tr>
<th>store_key</th>
<th>order_number</th>
<th>date_ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>202416</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>24</td>
<td>250295</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>121</td>
<td>251417</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>198</td>
<td>75716</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>166</td>
<td>36008</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>27</td>
<td>150241</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>148</td>
<td>182207</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>9</td>
<td>188567</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>113</td>
<td>66017</td>
<td>2013-01-04</td>
</tr>
</tbody>
</table>
```
vmart_query_07.sql

```
-- vmart_query_07.sql
-- Multicolumn subquery
-- Which products have the highest cost,
-- grouped by category and department
SELECT product_description, sku_number, department_description
FROM public.product_dimension
WHERE (category_description, department_description, product_cost) IN (  
    SELECT category_description, department_description,  
    MAX(product_cost) FROM product_dimension  
    GROUP BY category_description, department_description);
```

Output

<table>
<thead>
<tr>
<th>product_description</th>
<th>sku_number</th>
<th>department_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand #601 steak</td>
<td>SKU-#601</td>
<td>Meat</td>
</tr>
<tr>
<td>Brand #649 brooms</td>
<td>SKU-#649</td>
<td>Cleaning supplies</td>
</tr>
<tr>
<td>Brand #677 veal</td>
<td>SKU-#677</td>
<td>Meat</td>
</tr>
<tr>
<td>Brand #1371 memory card</td>
<td>SKU-#1371</td>
<td>Photography</td>
</tr>
<tr>
<td>Brand #1761 catfish</td>
<td>SKU-#1761</td>
<td>Seafood</td>
</tr>
<tr>
<td>Brand #1810 frozen pizza</td>
<td>SKU-#1810</td>
<td>Frozen Goods</td>
</tr>
<tr>
<td>Brand #1979 canned peaches</td>
<td>SKU-#1979</td>
<td>Canned Goods</td>
</tr>
<tr>
<td>Brand #2097 apples</td>
<td>SKU-#2097</td>
<td>Produce</td>
</tr>
<tr>
<td>Brand #2287 lens cap</td>
<td>SKU-#2287</td>
<td>Photography</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

vmart_query_08.sql

```
-- vmart_query_08.sql
-- Using pre-join projections to answer subqueries
-- between online_sales_fact and online_page_dimension
SELECT page_description, page_type, start_date, end_date
FROM online_sales.online_sales_fact f, online_sales.online_page_dimension d
WHERE f.online_page_key = d.online_page_key
AND page_number IN
    (SELECT MAX(page_number)
     FROM online_sales.online_page_dimension)
AND page_type = 'monthly' AND start_date = '2012-06-02';
```

Output

<table>
<thead>
<tr>
<th>page_description</th>
<th>page_type</th>
<th>start_date</th>
<th>end_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Page Description #1</td>
<td>monthly</td>
<td>2012-06-02</td>
<td>2012-06-11</td>
</tr>
<tr>
<td>Online Page Description #1</td>
<td>monthly</td>
<td>2012-06-02</td>
<td>2012-06-11</td>
</tr>
<tr>
<td>Online Page Description #1</td>
<td>monthly</td>
<td>2012-06-02</td>
<td>2012-06-11</td>
</tr>
<tr>
<td>Online Page Description #1</td>
<td>monthly</td>
<td>2012-06-02</td>
<td>2012-06-11</td>
</tr>
</tbody>
</table>
**vmart_query_09.sql**

```sql
-- vmart_query_09.sql
-- Equi join
-- Joins online_sales_fact table and the call_center_dimension
-- table with the ON clause

SELECT sales_quantity, sales_dollar_amount, transaction_type, cc_name
FROM online_sales.online_sales_fact
INNER JOIN online_sales.call_center_dimension
ON (online_sales.online_sales_fact.call_center_key
     = online_sales.call_center_dimension.call_center_key
AND sale_date_key = 156)
ORDER BY sales_dollar_amount DESC;
```

**Output**

<table>
<thead>
<tr>
<th>sales_quantity</th>
<th>sales_dollar_amount</th>
<th>transaction_type</th>
<th>cc_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>589</td>
<td>purchase</td>
<td>Central Midwest</td>
</tr>
<tr>
<td>8</td>
<td>589</td>
<td>purchase</td>
<td>South Midwest</td>
</tr>
<tr>
<td>8</td>
<td>589</td>
<td>purchase</td>
<td>California</td>
</tr>
<tr>
<td>1</td>
<td>587</td>
<td>purchase</td>
<td>New England</td>
</tr>
<tr>
<td>1</td>
<td>586</td>
<td>purchase</td>
<td>Other</td>
</tr>
<tr>
<td>1</td>
<td>584</td>
<td>purchase</td>
<td>New England</td>
</tr>
<tr>
<td>4</td>
<td>584</td>
<td>purchase</td>
<td>New England</td>
</tr>
<tr>
<td>7</td>
<td>581</td>
<td>purchase</td>
<td>Mid Atlantic</td>
</tr>
<tr>
<td>5</td>
<td>579</td>
<td>purchase</td>
<td>North Midwest</td>
</tr>
<tr>
<td>8</td>
<td>577</td>
<td>purchase</td>
<td>North Midwest</td>
</tr>
<tr>
<td>4</td>
<td>577</td>
<td>purchase</td>
<td>Central Midwest</td>
</tr>
<tr>
<td>2</td>
<td>575</td>
<td>purchase</td>
<td>Hawaii/Alaska</td>
</tr>
<tr>
<td>4</td>
<td>573</td>
<td>purchase</td>
<td>NY Metro</td>
</tr>
<tr>
<td>4</td>
<td>572</td>
<td>purchase</td>
<td>Central Midwest</td>
</tr>
<tr>
<td>1</td>
<td>570</td>
<td>purchase</td>
<td>Mid Atlantic</td>
</tr>
<tr>
<td>9</td>
<td>569</td>
<td>purchase</td>
<td>Southeaster</td>
</tr>
<tr>
<td>1</td>
<td>569</td>
<td>purchase</td>
<td>NY Metro</td>
</tr>
<tr>
<td>5</td>
<td>567</td>
<td>purchase</td>
<td>Other</td>
</tr>
<tr>
<td>7</td>
<td>567</td>
<td>purchase</td>
<td>Hawaii/Alaska</td>
</tr>
<tr>
<td>9</td>
<td>567</td>
<td>purchase</td>
<td>South Midwest</td>
</tr>
<tr>
<td>1</td>
<td>566</td>
<td>purchase</td>
<td>New England</td>
</tr>
</tbody>
</table>

...
Welcome to the Vertica Administrator's Guide. This document describes how to set up and maintain a Vertica Analytics Platform database.
Prerequisites

This document makes the following assumptions:

- You are familiar with the concepts discussed in the Vertica Concepts.
- Performed the following procedures as described in the Installing Vertica:
  - Constructed a hardware platform.
  - Installed Linux.
  - Installed Vertica and configured a cluster of hosts.
Administration Overview

This document describes the functions performed by a Vertica database administrator (DBA). Perform these tasks using only the dedicated database administrator account that was created when you installed Vertica. The examples in this documentation set assume that the administrative account name is dbadmin.

- To perform certain cluster configuration and administration tasks, the DBA (users of the administrative account) must be able to supply the root password for those hosts. If this requirement conflicts with your organization's security policies, these functions must be performed by your IT staff.

- If you perform administrative functions using a different account from the account provided during installation, Vertica encounters file ownership problems.

- If you share the administrative account password, make sure that only one user runs the Administration Tools at any time. Otherwise, automatic configuration propagation does not work correctly.

- The Administration Tools require that the calling user's shell be /bin/bash. Other shells give unexpected results and are not supported.
Managing Licenses

You must license Vertica in order to use it. Vertica supplies your license in the form of one or more license files, which encode the terms of your license.

To prevent introducing special characters that invalidate the license, do not open the license files in an editor. Opening the file in this way can introduce special characters, such as line endings and file terminators, that may not be visible within the editor. Whether visible or not, these characters invalidate the license.

Applying License Files

Be careful not to change the license key file in any way when copying the file between Windows and Linux, or to any other location. To help prevent applications from trying to alter the file, enclose the license file in an archive file (such as a .zip or .tar file). You should keep a back up of your license key file. OpenText recommends that you keep the backup in /opt/vertica.

After copying the license file from one location to another, check that the copied file size is identical to that of the one you received from Vertica.

Obtaining a License Key File

Important: If you are using Vertica 8.1.x or earlier, you will receive a temporary license key. Within two business days, you should receive an email containing your permanent license key file.

Follow these steps to obtain a license key file:

1. Log in to the Software Entitlement Key site using your passport login information. If you do not have a passport login, create one.

2. On the Request Access page, enter your order number and select a role.

3. Enter your request access reasoning.

4. Click Submit.
5. After your request is approved, you will receive a confirmation email. On the site, click the Entitlements tab to see your Vertica software.

6. Under the Action tab, click Activate. You may select more than one product.

7. The License Activation page opens. Enter your Target Name.

8. Select your Vertica version and the quantity you want to activate.

9. Click Next.

10. Confirm your activation details and click Submit.

11. The Activation Results page displays. Follow the instructions in New Vertica License Installations or Vertica License Changes to complete your installation or upgrade.

Your Vertica Community Edition download package includes the Community Edition license, which allows three nodes and 1TB of data. The Vertica Community Edition license does not expire.

Understanding Vertica Licenses

Vertica has flexible licensing terms. It can be licensed on the following bases:

- Term-based (valid until a specific date)
- Size-based (valid to store up to a specified amount of raw data)
- Both term- and size-based
- Unlimited duration and data storage
- Node-based with an unlimited number of CPUs and users (one node is a server acting as a single computer system, whether physical or virtual)

Vertica Community Edition licenses include 1 terabyte data and a limit of 3 nodes.

Vertica for SQL on Apache Hadoop is a separate product with its own license. This documentation covers both products. Consult your license agreement for details about available features and limitations.

Your license key has your licensing bases encoded into it. If you are unsure of your current license, you can view your license information from within Vertica.
Vertica Analytics Platform License Types

Vertica is a full-featured offering with all analytical functions described in this documentation. It is best used for advanced analytics and enterprise data warehousing.

Vertica Community Edition: You can download and start using Community Edition for free. The Community Edition license allows customers the following:

- 3 node limit
- 1 terabyte data limit

Community Edition licenses cannot be installed co-located in a Hadoop infrastructure and used to query data stored in Hadoop formats.

Vertica Premium Edition: You can purchase the Premium Edition license. The Premium Edition license entitles customers to:

- No node limit
- Data amount as specified by the license
- Store data in HDFS and read limited data from HDFS, as specified in your license

Premium Edition licenses cannot be installed co-located in a Hadoop infrastructure.

Note: Vertica does not support license downgrades.

Vertica for SQL on Apache Hadoop License

Vertica for SQL on Apache Hadoop is a license for running Vertica on a Hadoop environment. This allows users to run Vertica on data that is in a shared storage environment. It is best used for exploring data in a Hadoop data lake. It can be used only in co-located Hadoop environments to query data stored in Hadoop (Hortonworks, MapR, or Cloudera).

Customers can purchase this term-based SQL on Apache Hadoop license per the number of nodes they plan to use in their Hadoop environment. The license then audits the number of nodes being used for compliance.
Installing or Upgrading a License Key

The steps you follow to apply your Vertica license key vary, depending on the type of license you are applying and whether you are upgrading your license. This section describes the following:

- New Vertica License Installations
- Vertica License Renewals or Upgrades

New Vertica License Installations

Follow these steps to install a new Vertica license:

1. Copy the license key file you generated from the Software Entitlement Key site to your Administration Host.

2. Ensure the license key's file permissions are set to 400 (read permissions).

3. Install Vertica as described in the Installing Vertica if you have not already done so. The interface prompts you for the license key file.

4. To install Community Edition, leave the default path blank and click OK. To apply your evaluation or Premium Edition license, enter the absolute path of the license key file you downloaded to your Administration Host and press OK. The first time you log in as the Database Administrator and run the Administration Tools, the interface prompts you to accept the End-User License Agreement (EULA).

   **Note:** If you installed Management Console, the MC administrator can point to the location of the license key during Management Console configuration.

5. Choose View EULA.

6. Exit the EULA and choose Accept EULA to officially accept the EULA and continue installing the license, or choose Reject EULA to reject the EULA and return to the Advanced Menu.
Vertica License Changes

If your license is expiring or you want your database to grow beyond your licensed data size, you must renew or upgrade your license. After you obtain your renewal or upgraded license key file, you can install it using Administration Tools or Management Console.

Upgrading does not require a new license unless you are increasing the capacity of your database. As of Vertica 9.0, you can add-on capacity to your database using the Software Entitlement Key. You do not need uninstall and reinstall the license to add-on capacity. Add on licenses are not supported with Vertica 9.0 and later.

Uploading or Upgrading a License Key Using Administration Tools

1. Copy the license key file you generated from the Software Entitlement Key site to your Administration Host.

2. Ensure the license key's file permissions are set to 400 (read permissions).

3. Start your database, if it is not already running.

4. In the Administration Tools, select Advanced > Upgrade License Key and click OK.

5. Enter the absolute path to your new license key file and click OK. The interface prompts you to accept the End-User License Agreement (EULA).

6. Choose View EULA.

7. Exit the EULA and choose Accept EULA to officially accept the EULA and continue installing the license, or choose Reject EULA to reject the EULA and return to the Advanced Tools menu.

Uploading or Upgrading a License Key Using Management Console

1. From your database's Overview page in Management Console, click the License tab. The License page displays. You can view your installed licenses on this page.

2. Click Install New License at the top of the License page.
3. Browse to the location of the license key from your local computer and upload the file.

4. Click Apply at the top of the page. Management Console prompts you to accept the End-User License Agreement (EULA).

5. Select the check box to officially accept the EULA and continue installing the license, or click Cancel to exit.

Note: As soon as you renew or upgrade your license key from either your Administration Host or Management Console, Vertica applies the license update. No further warnings appear.

Adding Capacity

If you are adding capacity to your database, you do not need to uninstall and reinstall the license. Instead, you can install multiple licenses to increase the size of your database. This additive capacity only works for licenses with the same format, such as adding a Premium license capacity to an existing Premium license type. When you add capacity, the size of license will be the total of both licenses; the previous license is not overwritten. You cannot add capacity using two different license formats, such as adding Hadoop license capacity to an existing Premium license.

You can run the AUDIT() function to verify the license capacity was added on. The reflection of add-on capacity to your license will run during the automatic run of the audit function. If you want to see the immediate result of the add-on capacity, run the AUDIT() function to refresh.

Note: If you have an expired license, you must drop the expired license before you can continue to use Vertica. For more information, see DROP_LICENSE.

Viewing Your License Status

You can use several functions to display your license terms and current status.
Examining Your License Key

Use the DISPLAY_LICENSE SQL function described in the SQL Reference Manual to display the license information. This function displays the dates for which your license is valid (or Perpetual if your license does not expire) and any raw data allowance. For example:

```sql
=> SELECT DISPLAY_LICENSE();
DISPLAY_LICENSE
--------------------------------------
Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB
(1 row)
```

You can also query the LICENSES system table to view information about your installed licenses. This table displays your license types, the dates for which your licenses are valid, and the size and node limits your licenses impose.

Alternatively, use the LICENSES table in Management Console. On your database Overview page, click the License tab to view information about your installed licenses.

Viewing Your License Compliance

If your license includes a raw data size allowance, Vertica periodically audits your database's size to ensure it remains compliant with the license agreement. If your license has a term limit, Vertica also periodically checks to see if the license has expired. You can see the result of the latest audits using the GET_COMPLIANCE_STATUS function.

```sql
=> select GET_COMPLIANCE_STATUS();
GET_COMPLIANCE_STATUS
--------------------------------------
Raw Data Size: 2.000GB +/- 0.003GB
License Size : 4.000GB
Utilization : 50%
Audit Time : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

To see how your ORC/Parquet data is affecting your license compliance, see Viewing License Compliance for Hadoop File Formats.
Viewing Your License Status Through MC

Information about license usage is on the Settings page. See Monitoring Database Size for License Compliance.

Viewing License Compliance for Hadoop File Formats

You can use the EXTERNAL_TABLEDETAILS system table to gather information about your use of Hadoop file formats. This information can help you assess your compliance with your Vertica license.

Vertica computes the values in this table at query time, so to avoid performance problems, restrict your queries to filter by table_schema, table_name, or source_format. These three columns are the only columns you can use in a predicate, but you may use all of the usual predicate operators.

```
=> SELECT * FROM EXTERNAL_TABLEDETAILS
  WHERE source_format = 'PARQUET' OR source_format = 'ORC';
```

```
-[- RECORD 1 ]-+-------------------------------------------------------------+-
| schema_oid | 45035996273704978 |
| table_schema | public |
| table_oid | 45035996273760390 |
| table_name | ORC_demo |
| source_format | ORC |
| total_file_count | 5 |
| total_file_size_bytes | 789 |
| source_statement | COPY FROM 'ORC_demo/*' ORC |
| file_access_error | |

-[- RECORD 2 ]-+-------------------------------------------------------------+-
| schema_oid | 45035106277204374 |
| table_schema | public |
| table_oid | 45035996274460352 |
| table_name | Parquet_demo |
| source_format | PARQUET |
| total_file_count | 3 |
| total_file_size_bytes | 498 |
| source_statement | COPY FROM 'Parquet_demo/*' PARQUET |
```
When computing the size of an external table, Vertica counts all data found in the location specified by the COPY FROM clause. If you have a directory that contains ORC and delimited files, for example, and you define your external table with "COPY FROM *" instead of "COPY FROM *.orc", this table includes the size of the delimited files. (You would probably also encounter errors when querying that external table.) When you query this table Vertica does not validate your table definition; it just uses the path to find files to report.

Auditing Database Size

You can use your Vertica software until columnar data reaches the maximum raw data size that the your license agreement allows. Vertica periodically runs an audit of the columnar data size to verify that your database complies with this agreement. You can also run your own audits of database size with two functions:

- **AUDIT**: Estimates the raw data size of a database, schema, or table.
- **AUDIT_FLEX**: Estimates the size of one or more flexible tables in a database, schema, or projection.

The following two examples audit the database and schema online_sales:

```
=> SELECT AUDIT('', 'database');
AUDIT
----------
76376696
(1 row)
```

```
=> SELECT AUDIT('online_sales', 'schema');
AUDIT
----------
35716504
(1 row)
```

Raw Data Size

AUDIT and AUDIT_FLEX use statistical sampling to estimate the raw data size of the database—that is, the uncompressed data that the database stores. Vertica evaluates the raw data size as if the data were exported from the database in text format, rather than as compressed data.
By using statistical sampling, the audit minimizes its impact on database performance. The tradeoff between accuracy and performance impact is a small margin of error. Reports on your database size include the margin of error, so you can assess the accuracy of the estimate.

**Unaudited Data**

Table data that appears in multiple projections is counted only once. An audit also excludes the following data:

- Temporary table data.
- Data in `SET USING` columns.
- Data accessible through external table definitions.
- Data that was deleted but not yet purged.
- Data stored in the WOS.
- Data stored in system and work tables such as monitoring tables, Data Collector tables, and Database Designer tables.
- Delimiter characters.

**Evaluating Data Type Footprint**

Vertica evaluates the footprint of different data types as follows:

- Strings and binary types—`CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`—are counted as their actual size in bytes using UTF-8 encoding.

- Numeric data types are evaluated as if they were printed. Each digit counts as a byte, as does any decimal point, sign, or scientific notation. For example, `-123.456` counts as eight bytes—six digits plus the decimal point and minus sign.

- Date/time data types are evaluated as if they were converted to text, including hyphens, spaces, and colons. For example, `vsql` prints a timestamp value of `2011-07-04 12:00:00` as 19 characters, or 19 bytes.
Controlling Audit Accuracy

`AUDIT` can specify the level of an audit's error tolerance and confidence, by default set to 5 and 99 percent, respectively. For example, you can obtain a high level of audit accuracy by setting error tolerance and confidence level to 0 and 100 percent, respectively. Unlike estimating raw data size with statistical sampling, Vertica dumps all audited data to a raw format to calculate its size. If you audit the entire database at this level, the audit also includes contents of the WOS.

**Caution:** Vertica discourages database-wide audits at this level. Doing so can have a significant adverse impact on database performance.

The following example audits the database with 25% error tolerance:

```sql
=> SELECT AUDIT('', 25);
AUDIT
----------
75797126
(1 row)
```

The following example audits the database with 25% level of tolerance and 90% confidence level:

```sql
=> SELECT AUDIT('',25,90);
AUDIT
----------
76402672
(1 row)
```

Monitoring Database Size for License Compliance

Your Vertica license can include a data storage allowance. The allowance can consist of data in columnar tables, flex tables, or both types of data. The `AUDIT()` function estimates the columnar table data size and any flex table materialized columns. The `AUDIT_FLEX()` function estimates the amount of __raw__ column data in flex or columnar tables. In regards to license data limits, data in __raw__ columns is calculated at 1/10th the size of structured data. Monitoring data sizes for columnar and flex tables lets you plan either to schedule deleting old data to keep your database in compliance with your license, or to consider a license upgrade for additional data storage.
Note: An audit of columnar data includes flex table real and materialized columns, but not __raw__ column data.

Viewing Your License Compliance Status

Vertica periodically runs an audit of the columnar data size to verify that your database is compliant with your license terms. You can view the results of the most recent audit by calling the GET_COMPLIANCE_STATUS function.

```sql
=> select GET_COMPLIANCE_STATUS();
```

- Raw Data Size: 2.00GB +/- 0.003GB
- License Size: 4.000GB
- Utilization: 50%
- Audit Time: 2011-03-09 09:54:09.538704+00
- Compliance Status: The database is in compliance with respect to raw data size.
- License End Date: 04/06/2011
- Days Remaining: 28.59

(1 row)

Periodically running GET_COMPLIANCE_STATUS to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your database begins to near its columnar data allowance, you can use the other auditing functions described below to determine where your database is growing and how recent deletes affect the database size.

Manually Auditing Columnar Data Usage

You can manually check license compliance for all columnar data in your database using the AUDIT_LICENSE_SIZE function. This function performs the same audit that Vertica periodically performs automatically. The AUDIT_LICENSE_SIZE check runs in the background, so the function returns immediately. You can then query the results using GET_COMPLIANCE_STATUS.

Note: When you audit columnar data, the results include any flex table real and materialized columns, but not data in the __raw__ column. Materialized columns are virtual columns that you have promoted to real columns. Columns that you define when creating a flex table, or which you add with ALTER TABLE...ADD COLUMN statements are real columns. All __raw__ columns are real columns. However, since they consist of unstructured or semi-structured data, they are audited separately.
An alternative to AUDIT_LICENSE_SIZE is to use the AUDIT function to audit the size of the columnar tables in your entire database by passing an empty string to the function. This function operates synchronously, returning when it has estimated the size of the database.

```sql
=> SELECT AUDIT('');
AUDIT
---------
76376696
(1 row)
```

The size of the database is reported in bytes. The AUDIT function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the AUDIT function in the SQL Reference Manual for full details. Vertica does not count the AUDIT function results as an official audit. It takes no license compliance actions based on the results.

Note: The results of the AUDIT function do not include flex table data in __raw__ columns. Use the AUDIT_FLEX function to monitor data usage flex tables.

### Manually Auditing __raw__ Column Data

You can use the AUDIT_FLEX function to manually audit data usage for flex or columnar tables with a __raw__ column. The function calculates the encoded, compressed data stored in ROS containers for any __raw__ columns. Materialized columns in flex tables are calculated by the AUDIT function. The AUDIT_FLEX results do not include data in the __raw__ columns of temporary flex tables.

### Targeted Auditing

If audits determine that the columnar table estimates are unexpectedly large, consider schemas, tables, or partitions that are using the most storage. You can use the AUDIT function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the online_sales schema in the VMart example database, run the following command:

```sql
=> SELECT AUDIT('online_sales');
AUDIT
--------
35716504
(1 row)
```

You can also change the granularity of an audit to report the size of each object in a larger entity (for example, each table in a schema) by using the granularity argument of the AUDIT function. See the AUDIT function in the SQL Reference Manual.
Using Management Console to Monitor License Compliance

You can also get information about data storage of columnar data (for columnar tables and for materialized columns in flex tables) through the Management Console. This information is available in the database Overview page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.
- The Audit button returns the same information as the AUDIT() function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.

Managing License Warnings and Limits

Term License Warnings and Expiration

The term portion of a Vertica license is easy to manage—you are licensed to use Vertica until a specific date. If the term of your license expires, Vertica alerts you with messages appearing in the Administration Tools and vsq1. For example:

```
=> CREATE TABLE T (A INT);
NOTICE: Vertica license is in its grace period
HINT: Renew at http://www.vertica.com/
CREATE TABLE
```

Contact Vertica at http://www.vertica.com/about/contact-us/ as soon as possible to renew your license, and then install the new license. After the grace period expires, Vertica stops processing queries.

Note: If you have an expired license, you must drop the expired license before you can continue to use Vertica. For more information, see DROP_LICENSE.
Data Size License Warnings and Remedies

If your Vertica columnar license includes a raw data size allowance, Vertica periodically audits the size of your database to ensure it remains compliant with the license agreement. For details of this audit, see Auditing Database Size. You should also monitor your database size to know when it will approach licensed usage. Monitoring the database size helps you plan to either upgrade your license to allow for continued database growth or delete data from the database so you remain compliant with your license. See Monitoring Database Size for License Compliance for details.

If your database's size approaches your licensed usage allowance (above 75% of license limits), you will see warnings in the Administration Tools, vsqI, and Management Console. You have two options to eliminate these warnings:

- Upgrade your license to a larger data size allowance.
- Delete data from your database to remain under your licensed raw data size allowance. The warnings disappear after Vertica's next audit of the database size shows that it is no longer close to or over the licensed amount. You can also manually run a database audit (see Monitoring Database Size for License Compliance for details).

If your database continues to grow after you receive warnings that its size is approaching your licensed size allowance, Vertica displays additional warnings in more parts of the system after a grace period passes.

If Your Vertica Premium Edition Database Size Exceeds Your Licensed Limits

If your Premium Edition database size exceeds your licensed data allowance, all successful queries from ODBC and JDBC clients return with a status of SUCCESS_WITH_INFO instead of the usual SUCCESS. The message sent with the results contains a warning about the database size. Your ODBC and JDBC clients should be prepared to handle these messages instead of assuming that successful requests always return SUCCESS.

Note: These warnings for Premium Edition are in addition to any warnings you see in Administration Tools, vsqI, and Management Console.
If Your Vertica Community Edition Database Size Exceeds 1 Terabyte

If your Community Edition database size exceeds the limit of 1 terabyte, you will no longer be able to load or modify data in your database. In addition, you will not be able to delete data from your database.

To bring your database under compliance, you can choose to:

- Drop database tables. You can also consider truncating a table or dropping a partition. See `TRUNCATE TABLE` or `DROP_PARTITIONS`.

- Upgrade to Vertica Premium Edition (or an evaluation license).

Exporting License Audit Results to CSV

You can use `admintools` to audit a database for license compliance and export the results in CSV format, as follows:

```
admintools -t license_audit [--password=password] --database=database [--file=csv-file] [--quiet]
```

where:

- `database` must be a running database. If the database is password protected, you must also supply the password.

- `--file` `csv-file` directs output to the specified file. If `csv-file` already exists, the tool returns an error message. If this option is unspecified, output is directed to `stdout`.

- `--quiet` specifies that the tool should run in quiet mode; if unspecified, status messages are sent to `stdout`.

Running the `license_audit` tool is equivalent to invoking the following SQL statements:

```
select audit('');
select audit_flex('');
select * from dc_features_used;
select * from v_catalog.licenseAudits;
select * from v_catalog.userAudits;
```

Audit results include the following information:
- Log of used Vertica features
- Estimated database size
- Raw data size allowed by your Vertica license
- Percentage of licensed allowance that the database currently uses
- Audit timestamps

The following truncated example shows the raw CSV output that license_audit generates:

```plaintext
FEATURES_USED
features_used,feature,date,sum
features_used,metafunction::get_compliance_status,2014-08-04,1
features_used,metafunction::bootstrap_license,2014-08-04,1
...

LICENSE_AUDITS
license_audits,database_size_bytes,license_size_bytes,usage_percent,audit_start_timestamp,audit_end_timestamp,confidence_level_percent,error_tolerance_percent,used_sampling,confidence_interval_lower_bound_bytes,confidence_interval_upper_bound_bytes,sample_count,cell_count,license_name
license_audits,808117909,536870912000,0.00150523690320551,2014-08-04 23:59:00.024874-04,2014-08-04 23:59:00.578419-04,99,5,t,785472097,830763721,10000,174754646,vertica
...

USER_AUDITS
user_audits,size_bytes,user_id,user_name,object_id,object_type,object_schema,object_name,audit_start_timestamp,audit_end_timestamp,confidence_level_percent,error_tolerance_percent,used_sampling,confidence_interval_lower_bound_bytes,confidence_interval_upper_bound_bytes,sample_count,cell_count

AUDIT_SIZE_BYTES
audit_size_bytes,now,audit
audit_size_bytes,2014-10-14 11:52:14.015231-04,810584417

FLEX_SIZE_BYTES
flex_size_bytes,now,audit
flex_size_bytes,2014-10-14 11:52:15.117036-04,11850
```
Configuring the Database

This section provides information about:

- The Configuration Procedure
- Configuration Parameters
- Designing a logical schema
- Creating the physical schema

You'll also want to set up a security scheme. See Implementing Security.

See also implementing locales for international data sets.

Note: Vertica assumes that you are familiar with the material in Getting Started and are familiar with creating and configuring a fully-functioning example database.
Configuration Procedure

This section describes the tasks required to set up a Vertica database. It assumes that you have a valid license key file, installed the Vertica rpm package, and ran the installation script as described in Installing Vertica.

You complete the configuration procedure using:

- Administration Tools
  
  If you are unfamiliar with Dialog-based user interfaces, read Using the Administration Tools Interface before you begin. See also the Administration Tools Reference for details.

- vsql interactive interface

- Database Designer, described in Creating a Database Design

Note: You can also perform certain tasks using Management Console. Those tasks point to the appropriate topic.

Continuing Configuring

Follow the configuration procedure sequentially as this section describes.

Vertica strongly recommends that you first experiment with creating and configuring a database.

You can use this generic configuration procedure several times during the development process, modifying it to fit your changing goals. You can omit steps such as preparing actual data files and sample queries, and run the Database Designer without optimizing for queries. For example, you can create, load, and query a database several times for development and testing purposes, then one final time to create and load the production database.
Prepare Disk Storage Locations

You must create and specify directories in which to store your catalog and data files (physical schema). You can specify these locations when you install or configure the database, or later during database operations. Both the catalog and data directories must be owned by the database administrator.

The directory you specify for database catalog files (the catalog path) is used across all nodes in the cluster. For example, if you specify /home/catalog as the catalog directory, Vertica uses that catalog path on all nodes. The catalog directory should always be separate from any data file directories.

**Note:** Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

The data path you designate is also used across all nodes in the cluster. Specifying that data should be stored in /home/data, Vertica uses this path on all database nodes.

Do not use a single directory to contain both catalog and data files. You can store the catalog and data directories on different drives, which can be either on drives local to the host (recommended for the catalog directory) or on a shared storage location, such as an external disk enclosure or a SAN.

Before you specify a catalog or data path, be sure the parent directory exists on all nodes of your database. Creating a database in admintools also creates the catalog and data directories, but the parent directory must exist on each node.

You do not need to specify a disk storage location during installation. However, you can do so by using the --data-dir parameter to the install_vertica script. See [Specifying Disk Storage Location During Installation](#).

Specifying Disk Storage Location During Installation

You can specify the disk storage location when you:

- Install Vertica (see below).

- Create a database using the Administration Tools.
Install and configure Management Console.

Specifying Disk Storage Location When You Install

When you install Vertica, the --data-dir parameter in the install_vertica script lets you specify a directory to contain database data and catalog files. The script defaults to the database administrator's default home directory /home/dbadmin.

**Important:** Replace this default with a directory that has adequate space to hold your data and catalog files.

Requirements

- The data and catalog directory must exist on each node in the cluster.
- The directory on each node must be owned by the database administrator.
- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up. See Managing Storage Locations for details.

Specifying Disk Storage Location During Database Creation

When you invoke the Create Database command in the Administration Tools, a dialog box allows you to specify the catalog and data locations. These locations must exist on each host in the cluster and must be owned by the database administrator.

![Database data directories dialog box](image)

When you click OK, Vertica automatically creates the following subdirectories:
For example, if you use the default value (the database administrator's home directory) of 
/home/dbadmin for the Stock Exchange example database, the catalog and data directories
are created on each node in the cluster as follows:

/home/dbadmin/Stock_Schema/stock_schema_node1_host01_catalog/home/dbadmin/Stock_Schema/stock_schema_ node1_host01_data

Notes

- Catalog and data path names must contain only alphanumeric characters and cannot have
  leading space characters. Failure to comply with these restrictions will result in database
  creation failure.

- Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore,
  if you created a database for evaluation purposes, dropped the database, and
  want to reuse the database name, make sure that the disk storage location previously used
  has been completely cleaned up. See Managing Storage Locations for details.

Specifying Disk Storage Location on MC

You can use the MC interface to specify where you want to store database metadata on the
cluster in the following ways:

- When you configure MC the first time

- When you create new databases using on MC

See Also

Configuring Management Console.

Configuring Disk Usage to Optimize Performance

Once you have created your initial storage location, you can add additional storage locations to
the database later. Not only does this provide additional space, it lets you control disk usage
and increase I/O performance by isolating files that have different I/O or access patterns. For
example, consider:
Isolating execution engine temporary files from data files by creating a separate storage location for temp space.

Creating labeled storage locations and storage policies, in which selected database objects are stored on different storage locations based on measured performance statistics or predicted access patterns.

See Also

Managing Storage Locations

Using Shared Storage With Vertica

If using shared SAN storage, ensure there is no contention among the nodes for disk space or bandwidth.

- Each host must have its own catalog and data locations. Hosts cannot share catalog or data locations.

- Configure the storage so that there is enough I/O bandwidth for each node to access the storage independently.

Viewing Database Storage Information

You can view node-specific information on your Vertica cluster through the Management Console. See Monitoring Using MC for details.

Disk Space Requirements for Vertica

In addition to actual data stored in the database, Vertica requires disk space for several data reorganization operations, such as mergeout and managing nodes in the cluster. For best results, Vertica recommends that disk utilization per node be no more than sixty percent (60%) for a K-Safe=1 database to allow such operations to proceed.

In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they cannot be completed in memory (RAM). Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed (known as temp space) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By
default, any unused disk space on the data disk can be used as temp space. However, Vertica recommends provisioning temp space separate from data disk space.

See Also

Configuring Disk Usage to Optimize Performance.

Disk Space Requirements for Management Console

You can install Management Console on any node in the cluster, so it has no special disk requirements, other than disk space you allocate for your database cluster.

See Also

Disk Space Requirements for Vertica.

Prepare the Logical Schema Script

Designing a logical schema for a Vertica database is no different from designing one for any other SQL database. Details are described more fully in Designing a Logical Schema.

To create your logical schema, prepare a SQL script (plain text file, typically with an extension of .sql) that:

1. Creates additional schemas (as necessary). See Using Multiple Schemas.

2. Creates the tables and column constraints in your database using the CREATE TABLE command.

3. Defines the necessary table constraints using the ALTER TABLE command.

4. Defines any views on the table using the CREATE VIEW command.

You can generate a script file using:

- A schema designer application.
- A schema extracted from an existing database.
- A text editor.
One of the example database example-name_define_schema.sql scripts as a template. (See the example database directories in /opt/vertica/examples.)

In your script file, make sure that:

- Each statement ends with a semicolon.
- You use data types supported by Vertica, as described in the SQL Reference Manual.

Once you have created a database, you can test your schema script by executing it as described in Create the Logical Schema. If you encounter errors, drop all tables, correct the errors, and run the script again.

**Prepare Data Files**

Prepare two sets of data files:

- Test data files. Use test files to test the database after the partial data load. If possible, use part of the actual data files to prepare the test data files.

- Actual data files. Once the database has been tested and optimized, use your data files for your initial Bulk-Loading Data.

**How to Name Data Files**

Name each data file to match the corresponding table in the logical schema. Case does not matter.

Use the extension .tbl or whatever you prefer. For example, if a table is named Stock_Dimension, name the corresponding data file stock_dimension.tbl. When using multiple data files, append _nnn (where nnn is a positive integer in the range 001 to 999) to the file name. For example, stock_dimension.tbl_001, stock_dimension.tbl_002, and so on.

**Prepare Load Scripts**

Note: You can postpone this step if your goal is to test a logical schema design for validity.

Prepare SQL scripts to load data directly into physical storage using the COPY...DIRECT statement using vsql, or through ODBC as described in Connecting to Vertica.

You need scripts that load the:
Large tables

Small tables

Vertica recommends that you load large tables using multiple files. To test the load process, use files of 10GB to 50GB in size. This size provides several advantages:

- You can use one of the data files as a sample data file for the Database Designer.
- You can load just enough data to Perform a Partial Data Load before you load the remainder.
- If a single load fails and rolls back, you do not lose an excessive amount of time.
- Once the load process is tested, for multi-terabyte tables, break up the full load in file sizes of 250–500GB.

See Also

- Bulk-Loading Data
- Bulk-Loading Data
- Using Load Scripts
- Using Parallel Load Streams
- Enforcing Constraints
- About Load Errors

Tip: You can use the load scripts included in the example databases in Getting Started as templates.

Create an Optional Sample Query Script

The purpose of a sample query script is to test your schema and load scripts for errors.

Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.
Create an Empty Database

Two options are available for creating an empty database:

- Using the Management Console
- Using Administration Tools

Although you can create more than one database (for example, one for production and one for testing), there can be only one active database for each installation of Vertica Analytic Database.

Creating a Database Name and Password

Database Names

Database names must conform to the following rules:

- Be between 1-30 characters
- Begin with a letter
- Follow with any combination of letters (upper and lowercase), numbers, and/or underscores.

Database names are case sensitive; however, Vertica strongly recommends that you do not create databases with names that differ only in case. For example, do not create a database called mydatabase and another called MyDataBase.

Database Passwords

Database passwords can contain letters, digits, and special characters listed in the next table. Passwords cannot include non-ASCII Unicode characters.

The allowed password length is between 0-100 characters. The database superuser can change a Vertica user’s maximum password length using `ALTER PROFILE`.

You use Profiles to specify and control password definitions. For instance, a profile can define the maximum length, reuse time, and the minimum number or required digits for a password, as well as other details.
The following table lists special (ASCII) characters that Vertica permits in database passwords. Special characters can appear anywhere in a password string. For example, mypas$word or $mypasswordare all valid.

Caution: Using special characters outside of the ones listed below could cause database instability.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>pound sign</td>
</tr>
<tr>
<td>!</td>
<td>exclamation point</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>?</td>
<td>question mark</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>.</td>
<td>period</td>
</tr>
<tr>
<td>/</td>
<td>forward slash</td>
</tr>
<tr>
<td>=</td>
<td>equals sign</td>
</tr>
<tr>
<td>~</td>
<td>tilde</td>
</tr>
<tr>
<td>-</td>
<td>minus sign</td>
</tr>
<tr>
<td>$</td>
<td>dollar sign</td>
</tr>
<tr>
<td>_</td>
<td>underscore</td>
</tr>
<tr>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td></td>
<td>space</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>%</td>
<td>percent sign</td>
</tr>
<tr>
<td>&amp;</td>
<td>ampersand</td>
</tr>
</tbody>
</table>
Create an Empty Database Using MC

You can create a new database on an existing Vertica cluster through the Management Console interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign
back in to MC later; the creation process continues unless an unexpected error occurs. See the Notes section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in Configuration Procedure. You should also run the Database Designer, which you access through the Administration Tools, to create either a comprehensive or incremental design. Consider using the Tutorial in Getting Started to create a sample database you can start monitoring immediately.

How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the Databases and Clusters page, skip to the next step; otherwise:
   a. Connect to MC and sign in as an MC administrator.
   b. On the Home page, click Existing Infrastructure to view the Databases and Clusters page.

2. If no databases exist on the cluster, continue to the next step; otherwise:
   a. If a database is running on the cluster on which you want to add a new database, select the database and click Stop.
   b. Wait for the running database to have a status of Stopped.

3. Click the cluster on which you want to create the new database and click Create Database.

4. The Create Database wizard opens. Provide the following information:
   - Database name and password. See Creating a Database Name and Password for rules.
   - Optionally click Advanced to open the advanced settings and change the port, catalog path, and data path. By default the MC application/web server port is 5450 and paths are /home/dbadmin, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the install_vertica script. Do not use the default agent port 5444 as a new setting for the MC port. See MC Settings > Configuration for port values.

5. Click Continue.

6. Select nodes to include in the database.

   The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node
and clear Include host in new database to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the Include check box.

7. Click Create in the Database Configuration window to create the database on the nodes.

The creation process takes a few moments, after which the database starts and a Success message appears on the interface.

8. Click OK to close the success message.

The Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your Vertica cluster, which are not shown in standby mode, you must run the install_vertica script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
  - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database Message Center from the MC Home page.
  - Failure messages display on the database Manage page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.
  - To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the Node Details page.

- To create MC users and grant them access to an MC-managed database, see About MC Users and Creating an MC User.

See Also

- Creating a Cluster Using MC
- Troubleshooting with MC Diagnostics
- Restarting MC
Create a Database Using Administration Tools

1. Run the Administration Tools from your Administration Host as follows:

   ```
   $ /opt/vertica/bin/admintools
   ```

   If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, see Notes for Remote Terminal Users.

2. Accept the license agreement and specify the location of your license file. For more information see Managing Licenses for more information.

   This step is necessary only if it is the first time you have run the Administration Tools

3. On the Main Menu, click Configuration Menu, and click OK.

4. On the Configuration Menu, click Create Database, and click OK.

5. Enter the name of the database and an optional comment, and click OK. See Creating a Database Name and Password for naming guidelines and restrictions.

6. Establish the superuser password for your database.

   - To provide a password enter the password and click OK. Confirm the password by entering it again, and then click OK.

   - If you don't want to provide the password, leave it blank and click OK. If you don't set a password, Vertica prompts you to verify that you truly do not want to establish a superuser password for this database. Click Yes to create the database without a password or No to establish the password.

   Caution: If you do not enter a password at this point, the superuser password is set to empty. Unless the database is for evaluation or academic purposes, Vertica strongly recommends that you enter a superuser password. See Creating a Database Name and Password for guidelines.

7. Select the hosts to include in the database from the list of hosts specified when Vertica was installed (install_vertica -s), and click OK.

8. Specify the directories in which to store the data and catalog files, and click OK.
Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

9. Catalog and data path names must contain only alphanumeric characters and cannot have leading spaces. Failure to comply with these restrictions results in database creation failure.

   For example:

   Catalog pathname: /home/dbadmin
   Data Pathname: /home/dbadmin

10. Review the Current Database Definition screen to verify that it represents the database you want to create, and then click Yes to proceed or No to modify the database definition.

11. If you click Yes, Vertica creates the database you defined and then displays a message to indicate that the database was successfully created.

   Note: For databases created with 3 or more nodes, Vertica automatically sets K-safety to 1 to ensure that the database is fault tolerant in case a node fails. For more information, see Failure Recovery in the Administrator's Guide and MARK_DESIGN_KSAFE

12. Click OK to acknowledge the message.

Create the Logical Schema

1. Connect to the database.

   In the Administration Tools Main Menu, click Connect to Database and click OK.

   See Connecting to the Database for details.
The `vsql` welcome script appears:

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.
Type: \h or \? for help with vsql commands
     \g or terminate with semicolon to execute query
     \q to quit

=>
```

2. **Run the logical schema script**

   Using the `\i` meta-command in vsql to run the SQL logical schema script that you prepared earlier.

3. **Disconnect from the database**

   Use the `\q` meta-command in vsql to return to the Administration Tools.

### Perform a Partial Data Load

Vertica recommends that for large tables, you perform a partial data load and then test your database before completing a full data load. This load should load a representative amount of data.

1. **Load the small tables.**

   Load the small table data files using the SQL load scripts and data files you prepared earlier.

2. **Partially load the large tables.**
Load 10GB to 50GB of table data for each table using the SQL load scripts and data files that you prepared earlier.

For more information about projections, see Physical Schema in Vertica Concepts.

Test the Database

Test the database to verify that it is running as expected.

Check queries for syntax errors and execution times.

1. Use the vsq1 \timing meta-command to enable the display of query execution time in milliseconds.
2. Execute the SQL sample query script that you prepared earlier.
3. Execute several ad hoc queries.

Optimize Query Performance

Optimizing the database consists of optimizing for compression and tuning for queries. (See Creating a Database Design.)

To optimize the database, use the Database Designer to create and deploy a design for optimizing the database. See Using Database Designer to Create a Comprehensive Design in Getting Started.

After you run the Database Designer, use the techniques described in Query Optimization in Analyzing Data to improve the performance of certain types of queries.

**Note:** The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

Complete the Data Load

To complete the load:
1. Monitor system resource usage.

Continue to run the top, free, and df utilities and watch them while your load scripts are running (as described in Monitoring Linux Resource Usage). You can do this on any or all nodes in the cluster. Make sure that the system is not swapping excessively (watch kswapd in top) or running out of swap space (watch for a large amount of used swap space in free).

**Note:** Vertica requires a dedicated server. If your loader or other processes take up significant amounts of RAM, it can result in swapping.

2. Complete the large table loads.

Run the remainder of the large table load scripts.

**Test the Optimized Database**

Check query execution times to test your optimized design:

1. Use the `vsql \timing` meta-command to enable the display of query execution time in milliseconds.

Execute a SQL sample query script to test your schema and load scripts for errors.

**Note:** Include a sample of queries your users are likely to run against the database. If you don’t have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

2. Execute several ad hoc queries

   a. Run Administration Tools and select **Connect to Database**.

   b. Use the `\i meta-command` to execute the query script; for example:

```
vmartdb=> \i vmart_query_03.sql  customer_name | annual_income
---------------------------------------------
 James M. McNulty |  999979
 Emily G. Vogel   |  999998
(2 rows)
Time: First fetch (2 rows): 58.411 ms. All rows formatted: 58.448 ms

vmartdb=> \i vmart_query_06.sql
store_key | order_number | date_ordered
---------------------------------------------
```
Once the database is optimized, it should run queries efficiently. If you discover queries that you want to optimize, you can modify and update the design. See Incremental Design in the Administrator's Guide.

Set Up Incremental (Trickle) Loads

Once you have a working database, you can use trickle loading to load new data while concurrent queries are running.

Trickle load is accomplished by using the COPY command (without the DIRECT keyword) to load 10,000 to 100,000 rows per transaction into the WOS. This allows Vertica to batch multiple loads when it writes data to disk. While the COPY command defaults to loading into the WOS, it will write ROS if the WOS is full.

See Trickle Loading Data for details.

See Also

- COPY
- Loading Data Through ODBC
Implement Locales for International Data Sets

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate GROUP BY and ORDER BY clauses, joins, and the analytic ORDER BY clause.

The default locale for a Vertica database is `en_US@collation=binary` (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default `en_US@collation=binary` collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to `en_US_POSIX`. You can set the locale back to the default locale and collation by issuing the `vsql` meta-command `\locale`. For example:

```
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation-binary;
```

You can set locale through ODBC, JDBC, and ADO.net.

ICU Locale Support

Vertica uses the ICU library for locale support; you must specify locale using the ICU locale syntax. The locale used by the database session is not derived from the operating system (through the LANG variable), so Vertica recommends that you set the LANG for each node running vsql, as described in the next section.

While ICU library services can specify collation, currency, and calendar preferences, Vertica supports only the collation component. Any keywords not relating to collation are rejected. Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.
The SET DATESTYLE TO ... command provides some aspects of the calendar, but Vertica supports only dollars as currency.

Changing DB Locale for a Session

This examples sets the session locale to Thai.

1. At the operating-system level for each node running vsqI, set the LANG variable to the locale language as follows:

   ```
   export LANG=th_TH.UTF-8
   ```

   Note: If setting the LANG= as shown does not work, the operating system support for locales may not be installed.

2. For each Vertica session (from ODBC/JDBC or vsqI) set the language locale.

   From vsqI:

   ```
   \locale th_TH
   ```

3. From ODBC/JDBC:

   ```
   "SET LOCALE TO th_TH;"
   ```

4. In PUTTY (or ssh terminal), change the settings as follows:

   ```
   settings > window > translation > UTF-8
   ```

5. Click Apply and then click Save.

All data loaded must be in UTF-8 format, not an ISO format, as described in Loading Delimited Data. Character sets like ISO 8859-1 (Latin1), which are incompatible with UTF-8, are not supported, so functions like SUBSTRING do not work correctly for multibyte characters. Thus, settings for locale should not work correctly. If the translation setting ISO-8859-11:2001 (Latin/Thai) works, the data is loaded incorrectly. To convert data correctly, use a utility program such as Linux iconv.

Note: The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets (bytes) that can be stored in that field, not the number of characters. When using multi-byte UTF-8 characters, make sure to size fields to accommodate from 1...
to 4 bytes per character, depending on the data.

See Also

- Supported Locales
- About Locale
- SET LOCALE
- ICU User Guide

Specify the Default Locale for the Database

After you start the database, the default locale configuration parameter, DefaultSessionLocale, sets the initial locale. You can override this value for individual sessions.

To set the locale for the database, use the configuration parameter as follows:

```sql
=> ALTER DATABASE mydb SET DefaultSessionLocale = 'ICU-locale-identifier';
```

For example:

```sql
=> ALTER DATABASE mydb SET DefaultSessionLocale = 'en_GB';
```

Override the Default Locale for a Session

To override the default locale for a specific session, use one of the following commands:

- The `vsql` command

  ```sql
  \locale <ICU-locale-identifier>;
  ```

  For example:

  ```sql
  => \locale en_GBINFO:
  INFO 2567:  Canonical locale: 'en_GB'
  Standard collation: 'LEN'
  English (United Kingdom)
  ```

- The statement `SET LOCALE TO <ICU-locale-identifier>`.
You can also use the **Specifying Locale: Short Form** of a locale in either of these commands:

```plaintext
=> SET LOCALE TO en_GB;
INFO 2567: Canonical locale: 'en_GB'
Standard collation: 'LEN'
English (United Kingdom)
```

```plaintext
=> SET LOCALE TO LEN;
INFO 2567: Canonical locale: 'en'
Standard collation: 'LEN'
English
```

You can use these commands to override the locale as many times as needed during a database session. The session locale setting applies to any subsequent commands issued in the session.

**See Also**

- **SET LOCALE**

**Server versus Client Locale Settings**

Vertica differentiates database server locale settings from client application locale settings:

- Server locale settings only impact collation behavior for server-side query processing.
- Client applications verify that locale is set appropriately in order to display characters correctly.

The following sections describe best practices to ensure predictable results.

**Server Locale**

The server session locale should be set as described in **Specify the Default Locale for the Database**. If locales vary across different sessions, set the server locale at the start of each session from your client.
vsq1 Client

- If the database does not have a default session locale, set the server locale for the session to the desired locale.
- The locale setting in the terminal emulator where the vsq1 client runs should be set to be equivalent to session locale setting on the server side (ICU locale). By doing so, the data is collated correctly on the server and displayed correctly on the client.
- All input data for vsq1 should be in UTF-8, and all output data is encoded in UTF-8
- Vertica does not support non UTF-8 encodings and associated locale values.
- For instructions on setting locale and encoding, refer to your terminal emulator documentation.

ODBC Clients

- ODBC applications can be either in ANSI or Unicode mode. If the user application is Unicode, the encoding used by ODBC is UCS-2. If the user application is ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 used on the database server. The ODBC driver converts UCS-2 to UTF-8 when passing to the Vertica server and converts data sent by the Vertica server from UTF-8 to UCS-2.
- If the user application is not already in UCS-2, the application must convert the input data to UCS-2, or unexpected results could occur. For example:
  - For non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
  - The symbol provided in the alternate encoding could be a valid UCS-2 symbol. If this occurs, incorrect data is inserted into the database.
- If the database does not have a default session locale, ODBC applications should set the desired server session locale using SQLSetConnectAttr (if different from database wide setting). By doing so, you get the expected collation and string functions behavior on the server.
JDBC and ADO.NET Clients

- JDBC and ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions apply as for ODBC if this encoding is violated.

- The JDBC and ADO.NET drivers convert UTF-16 data to UTF-8 when passing to the Vertica server and convert data sent by Vertica server from UTF-8 to UTF-16.

- If there is no default session locale at the database level, JDBC and ADO.NET applications should set the correct session server locale by executing the `SET LOCALE TO` command in order to get the expected collation and string functions behavior on the server. For more information, see `SET LOCALE`.

Change Transaction Isolation Levels

By default, Vertica uses the READ COMMITTED isolation level for all sessions. You can change the default isolation level for the database or for a given session.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

Database Isolation Level

The configuration parameter `TransactionIsolationLevel` specifies the database isolation level, and is used as the default for all sessions. Use `ALTER DATABASE` to change the default isolation level. For example:

```sql
=> ALTER DATABASE mydb SET TransactionIsolationLevel = 'SERIALIZABLE';
ALTER DATABASE
=> ALTER DATABASE mydb SET TransactionIsolationLevel = 'READ COMMITTED';
ALTER DATABASE
```

Changes to the database isolation level only apply to future sessions. Existing sessions and their transactions continue to use their original isolation level.

Use `SHOW CURRENT` to view the database isolation level:

```sql
=> SHOW CURRENT TransactionIsolationLevel;
  level | name            | setting
```
Session Isolation Level

`SET SESSION CHARACTERISTICS AS TRANSACTION` changes the isolation level for a specific session. For example:

```sql
=> SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET
```

Use `SHOW` to view the current session's isolation level:

```sql
=> SHOW TRANSACTION_ISOLATION;
```

See Also

Transactions
Configuration Parameters

Configuration parameters are settings that affect database behavior. You can use configuration parameters to enable, disable, or tune features related to different database aspects like Tuple Mover, security, Database Designer, or projections. Configuration parameters have default values, stored in the Vertica database.

You can modify certain parameters to configure your Vertica database in two ways:

- Management Console browser-based interface
- VSQL statements

Before you modify a database parameter, review all documentation about the parameter to determine the context under which you can change it. Some parameter changes require a database restart to take effect. The CHANGE_REQUIRES_RESTART column in the system table CONFIGURATION_PARAMETERS indicates whether a parameter requires a restart.

Managing Configuration Parameters: Management Console

To change database settings for any MC-managed database, click the Settings tab at the bottom of the Overview, Activity, or Manage pages. The database must be running.

The Settings page defaults to parameters in the General category. To change other parameters, click an option from the tab panel on the left.
Some settings require you to restart the database, and MC prompts you to do so. You can ignore the prompt, but those changes take effect only after the database restarts.

Some settings are specific to Management Console, such as changing MC or agent port assignments.

**See Also**

Managing MC Settings

**Managing Configuration Parameters: VSQL**

You can configure all parameters at database scope. Some parameters can also be set and cleared at node and session scopes.

*Caution:* Vertica is designed to operate with minimal configuration changes. Be careful to set and change configuration parameters according to documented guidelines.

**Viewing Configuration Parameter Values**

You can view active configuration parameter values in two ways:
SHOW statements

Query related system tables

SHOW Statements

Use the following SHOW statements to view active configuration parameters:

- **SHOW CURRENT**: Returns settings of active configuration parameter values. Vertica checks settings at all levels, in the following ascending order of precedence:
  - session
  - node
  - database

  If no values are set at any scope, SHOW CURRENT returns the parameter’s default value.

- **SHOW DATABASE**: Displays configuration parameter values set for the database.

- **SHOW SESSION**: Displays configuration parameter values set for the current session.

- **SHOW NODE**: Displays configuration parameter values set for a node.

If a configuration parameter requires a restart to take effect, the values in a SHOW CURRENT statement might differ from values in other SHOW statements. To see which parameters require restart, query the CONFIGURATION_PARAMETERS system table.

System Tables

You can query two system tables for configuration parameters:

- **SESSION_PARAMETERS** returns session-scope parameters.

- **CONFIGURATION_PARAMETERS** returns parameters for all scopes: database, node, and session.

Setting Configuration Parameter Values

You can set configuration parameters at three scopes:
Database Scope

You can set one or more parameter values at the database scope with `ALTER DATABASE..SET`:

```
ALTER DATABASE dbname SET parameter-name = value[,...];
```

For example:

```
ALTER DATABASE mydb SET AnalyzeRowCountInterval = 3600, FailoverToStandbyAfter = '5 minutes';
```

Node Scope

You can set one or more parameter values at the node scope with `ALTER NODE..SET`:

```
ALTER NODE node-name SET parameter-name = value[,...];
```

For example, to prevent clients from connecting to `v_vmart_node0001`, set the `MaxClientSessions` configuration parameter to 0:

```
=> ALTER NODE v_vmart_node0001 SET MaxClientSessions = 0;
```

Session Scope

You can set one or more parameter values at the session scope with `ALTER SESSION..SET`:

```
ALTER SESSION SET parameter-name = value[,...];
```

For example:

```
=> ALTER SESSION SET ForceUDxFencedMode = 1;
```

Clearing Configuration Parameters

You can clear configuration parameter settings at three scopes:
• Database
• Node
• Session

Database Scope

`ALTER DATABASE .. CLEAR` clears one or more parameter values at the database scope, and resets them to their default values as follows:

```
ALTER DATABASE dbname CLEAR parameter-name[,...];
```

For example:

```
ALTER DATABASE mydb CLEAR AnalyzeRowCountInterval, FailoverToStandbyAfter;
```

Node Scope

`ALTER NODE .. CLEAR` clears one or more parameter values at the node scope, and resets them to their database settings, if any. If the parameters are not set at the database scope, Vertica resets them to their default value.

```
ALTER NODE node-name CLEAR parameter-name[,...];
```

The following example clears MaxClientSessions on node `v_vmart_node0001`:

```
ALTER NODE v_vmart_node0001 CLEAR MaxClientSessions;
```

Session Scope

`ALTER SESSION .. CLEAR` clears one or more parameter values at the session scope, and resets them to their node or database settings, if any. If the parameters are not set at either scope, Vertica resets them to their default value.

```
ALTER SESSION CLEAR parameter-name[,...];
```

For example:

```
=> ALTER SESSION CLEAR ForceUDxFencedMode;
```
## Configuration Parameter Categories

Vertica configuration parameters are grouped into the following categories.

### General Parameters

You use these general parameters to configure Vertica.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AnalyzeRowCountInterval</code></td>
<td>Specifies how often Vertica checks the number of projection rows and whether the threshold set by ARCCommitPercentage has been crossed. For more information, see Collecting Statistics.</td>
<td>60 seconds</td>
</tr>
<tr>
<td><code>ApplyEventsDuringSALCheck</code></td>
<td>When enabled, Vertica uses catalog events to filter out dropped corrupt partitions during node startup. Dropping corrupt partitions can speed node recovery. When disabled, Vertica reports corrupt partitions, but takes no action. Leaving corrupt partitions in place can reset the current projection checkpoint epoch to the epoch before the corruption occurred. This parameter has no effect on unpartitioned tables.</td>
<td>0</td>
</tr>
<tr>
<td><code>ApportionedFileMinimumPortionSizeKB</code></td>
<td>Specifies the minimum portion size (in kilobytes) for use with apportioned file loads. Vertica apportions a file load across multiple nodes only if:</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The load can be divided into portions at least equaling this value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• EnableApportionedFileLoad and EnableApportionLoad are set to 1 (enabled).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See also EnableApportionLoad and EnableApportionedFileLoad.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 1024</td>
<td></td>
</tr>
<tr>
<td><strong>ARCCommitPercentage</strong></td>
<td>Sets the threshold percentage of WOS to ROS rows, which determines when to aggregate projection row counts and commit the result to the catalog. Vertica performs this action when the WOS to ROS percentage exceeds this setting.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 3 (percent)</td>
<td></td>
</tr>
<tr>
<td><strong>BlockedSocketGracePeriod</strong></td>
<td>Sets how long a session socket remains blocked while awaiting client input or output for a given query.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> None (Socket blocking can continue indefinitely.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See <a href="#">Handling Session Socket Blocking</a>.</td>
<td></td>
</tr>
<tr>
<td><strong>CatalogCheckpointPercent</strong></td>
<td>Specifies the threshold at which a checkpoint is created for the database catalog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>By default, this parameter is set to 50 (percent), so when transaction logs reach 50% of the size of the last checkpoint, Vertica adds a checkpoint. Each checkpoint demarcates all changes to the catalog since the last checkpoint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 50 (percent)</td>
<td></td>
</tr>
<tr>
<td><strong>ClusterSequenceCacheMode</strong></td>
<td>Indicates whether the initiator node</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>requests cache for other nodes in a cluster, and then sends cache to other nodes along with the execution plan. Enabled by default. When disabled, all nodes request their own cache.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 1 (enabled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 1: Initiator node requests cache.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 0: All nodes request their own cache.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Distributing Named Sequences.</td>
<td></td>
</tr>
<tr>
<td>CompressCatalogOnDisk</td>
<td>Compresses the size of the catalog on disk when enabled (value set to 1 or 2).</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 1: Compress checkpoints, but not logs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 2: Compress checkpoints and logs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Consider enabling this parameter if the catalog disk partition is small (&lt;50 GB) and the metadata is large (hundreds of tables, partitions, or nodes).</td>
<td></td>
</tr>
<tr>
<td>CompressNetworkData</td>
<td>Compresses all data sent over the internal network when enabled (value set to 1). This compression speeds up network traffic at the expense of added CPU load.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the network is throttling database performance, enable compression to correct the issue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 0</td>
<td></td>
</tr>
<tr>
<td>CopyFaultTolerantExpressions</td>
<td>Indicates whether to report record rejections during transformations and proceed (true) or abort COPY operations if</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a transformation fails (false).</td>
<td>Default Value: 0 (false)</td>
<td></td>
</tr>
<tr>
<td>CopyFromVerticaWithIdentity</td>
<td>Allows COPY FROM VERTICA and EXPORT TO VERTICA to load values into Identity (or Auto-increment) columns. The destination Identity column is not incremented automatically. To disable the default behavior, set this parameter to 0 (zero). Default Value: 1</td>
<td></td>
</tr>
<tr>
<td>DatabaseHeartBeatInterval</td>
<td>Determines the interval (in seconds) at which each node performs a health check and communicates a heartbeat. If a node does not receive a message within five times of the specified interval, the node is evicted from the cluster. Setting the interval to 0 disables the feature. Default Value: 120 See Automatic Eviction of Unhealthy Nodes.</td>
<td></td>
</tr>
<tr>
<td>DivideZeroByZeroThrowsError</td>
<td>If set to 1, returns an error if a division by zero operation is requested. Otherwise, returns 0 as the result of a division by zero operation. Default Value: 1</td>
<td></td>
</tr>
<tr>
<td>EnableApportionedChunkingInDefaultLoadParser</td>
<td>Enables the built-in parser for delimited files to take advantage of both apportioned load and cooperative parse for potentially better performance. Default Value: 1</td>
<td></td>
</tr>
<tr>
<td>EnableApportionedFileLoad</td>
<td>Enables automatic apportioning across nodes of file loads using COPY FROM. Vertica attempts to apportion the load if:</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| EnableApportionLoad             | Enables automatic apportioning across nodes of data loads using COPY WITH SOURCE. Vertica attempts to apportion the load if:  
  - This parameter is enabled.  
  - The source and parser both support apportioning.  
  Setting this parameter does not guarantee that loads will be apportioned, but disabling it guarantees that they will not be.  
  **Default Value:** 1  
  See Using Parallel Load Streams.                                                                                                                                                                                                                                                                                                                                 |
| EnableBetterFlexTypeGuessing    | Enables more accurate type guessing when assigning data types to non-string keys in a flex table __raw__ column with COMPUTE_FLEXTABLE KEYS or COMPUTE_FLEXTABLE KEYS AND_...
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_VIEW</td>
<td>This option is on by default. Turning this option off uses a limited set of Vertica data type assignments.</td>
</tr>
<tr>
<td><strong>Default Value:</strong> 1</td>
<td>See Setting Flex Table Configuration Parameters.</td>
</tr>
<tr>
<td>EnableCooperativeParse</td>
<td>Implements multi-threaded parsing capabilities on a node. You can use this parameter for both delimited and fixed-width loads. Enabled by default.</td>
</tr>
<tr>
<td>EnableDataTargetParallelism</td>
<td>Enables multiple threads for sorting and writing data to ROS, improving data loading performance. Enabled by default.</td>
</tr>
<tr>
<td>EnableForceOuter</td>
<td>Determines whether Vertica uses a table's force_outer value to implement a join. For more information, see Controlling Join Inputs.</td>
</tr>
<tr>
<td>EnableMetadataMemoryTracking</td>
<td>Enables Vertica to track memory used by database metadata in the METADATA resource pool.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Parameter</td>
<td>available memory in the General Pool. Tracking the catalog size separately in the new METADATA resource pool can reduce amount of memory available in the General Pool and can affect this thread calculation.</td>
</tr>
<tr>
<td>EnableResourcePoolCPUAffinity</td>
<td>Aligns queries to the resource pool of the processing CPU. When disabled (value is set to 0), queries run on any CPU, regardless of the CPU_AFFINITY_SET of the resource pool. Enabled by default.</td>
</tr>
<tr>
<td>EnableStorageBundling</td>
<td>(Deprecated) Enables storing multiple ROS containers as a single file. Each ROS must be less than the size specified in MaxBundleableROSSizeKB. In environments with many small storage files, bundling improves the performance of any file-intensive operations, including backups, restores, mergeouts and moveouts.</td>
</tr>
<tr>
<td>EnableUniquenessOptimization</td>
<td>Enables query optimization that is based on guaranteed uniqueness of column values. Columns that can be guaranteed to include unique values include:</td>
</tr>
<tr>
<td></td>
<td>• Columns that are defined with AUTO_INCREMENT or IDENTITY constraints</td>
</tr>
<tr>
<td></td>
<td>• Primary key columns where key constraints are enforced</td>
</tr>
<tr>
<td></td>
<td>• Columns that are constrained to unique values, either individually or as</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>a set</td>
<td>Default Value: 1 (enabled)</td>
</tr>
<tr>
<td>EnableWithClauseMaterialization</td>
<td>Enables materialization of WITH clause results. When materialization is enabled, Vertica evaluates each WITH clause once and stores results in a temporary table. This parameter can only be set at session level. Default Value: 0 (disabled) See WITH Clauses in SELECT in Analyzing Data.</td>
</tr>
<tr>
<td>ExternalTablesExceptionsLimit</td>
<td>Determines the maximum number of COPY exceptions and rejections allowed when a SELECT statement references an external table. Set to -1 to remove any exceptions limit. See Validating External Tables. Default Value: 100</td>
</tr>
<tr>
<td>FailoverToStandbyAfter</td>
<td>Specifies the length of time that an active standby node waits before taking the place of a failed node. This parameter is set to an interval literal. Default Value: None</td>
</tr>
<tr>
<td>FencedUDxMemoryLimitMB</td>
<td>Sets the maximum amount of memory, in megabytes (MB), that a fenced-mode UDF can use. If a UDF attempts to allocate more memory than this limit, that attempt triggers an exception. For more information, see Fenced Mode in Extending Vertica. Default Value: -1 (no limit)</td>
</tr>
<tr>
<td>FlexTableDataTypeGuessMultiplier</td>
<td>Specifies a multiplier that the COMPUTE_</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>FLEXTABLE_KEYS and COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW functions use when assigning a data type and column width for the flex keys table. Both functions assign each key a data type, and multiply the longest key value by this factor to estimate column width. Default Value: 2.0: The column width multiplier. Must be a value in the following range: Any value that results in a column width neither less than 20 bytes nor greater than FlexTableRawSize. This range is a cap to round sizes up or down, accordingly. See Setting Flex Table Configuration Parameters.</td>
</tr>
<tr>
<td>FlexTableRawSize</td>
<td>Specifies the default column width for the <em><strong>raw</strong></em> column of new flex tables. Default Value: 130000 Value range: 1 – 32000000</td>
</tr>
<tr>
<td>JavaBinaryForUDx</td>
<td>Sets the full path to the Java executable that Vertica uses to run Java UDxs. See Installing Java on Vertica Hosts in Extending Vertica.</td>
</tr>
<tr>
<td>JavaClassPathForUDx</td>
<td>Sets the Java classpath for the JVM that executes Java UDxs. This parameter must list all directories containing JAR files that Java UDxs import. Default Value: ${vertica_home}/packages/hcat/lib/* See Handling Java UDx Dependencies in Extending Vertica.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| MaxAutoSegColumns            | Specifies the number of columns (0–1024) to segment automatically when creating auto-projections from COPY and INSERT INTO statements. Setting this parameter to zero (0) uses all columns in the hash segmentation expression.  
  **Default Value:** 32        |
| MaxBundleableROSSizeKB       | Specifies the minimum size, in kilobytes, of an independent ROS file. When EnableStorageBundling is true, Vertica bundles storage container ROS files below this size into a single file. Bundling improves the performance of any file-intensive operations, including backups, restores, mergeouts and moveouts.  
  If you enable storage bundling and specify this parameter with a value of 0, Vertica bundles .fdb and .pidx files without bundling other storage container files.  
  **Default Value:** 1024      |
| MaxClientSessions            | Determines the maximum number of client sessions that can run on a single node of the database. The default value allows for five additional administrative logins. These logins prevent DBAs from being locked out of the system if non-dbadmin users reach the login limit.  
  **Tip:** Setting this parameter to 0 prevents new client sessions from being opened while you are shutting down the database. Restore the parameter to its original setting after you restart the database. For details, see Managing Sessions.  
  **Default Value:** 50 user logins and 5 |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional administrative logins</td>
<td></td>
</tr>
<tr>
<td>MaxTieredPoolScale</td>
<td>Specifies the threshold for allocating chunks of memory from the system or dedicated pools for Vertica system components—for example, for WOS containers. Above this threshold, Vertica allocates memory from the system (malloc), and returns it when no longer needed. Below this threshold, Vertica allocates memory from dedicated pools (talloc), which manage smaller memory allocations more efficiently. Tiered memory management facilitates more efficient use of resources, and minimizes the risk of any one process (such as the database catalog) from preemptively allocating and retaining excessive amounts of memory. Set this parameter to the desired power of 2, any value $\geq 20$. For example, for 128 MB, set the parameter to 27 ($2^{27}$). This parameter cannot be set below 20 (1MB). Any allocation that is less than 20 always uses talloc, regardless of this parameter setting. This parameter must be set at the database level, and takes effect only on database restart. <strong>Default Value:</strong> 27 (128 MB)</td>
</tr>
<tr>
<td>PatternMatchAllocator</td>
<td>Overrides the heap memory allocator for the pattern-match library when set to 1. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect. <strong>Default Value:</strong> 0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Parameter Description</td>
<td>See Regular Expression Functions.</td>
</tr>
</tbody>
</table>
| PatternMatchingUseJit                         | Enables just-in-time compilation (to machine code) of regular expression pattern matching functions used in queries. Using this parameter can usually improve pattern matching performance on large tables. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect.  
**Default Value:** 1  
See Regular Expression Functions. |
| PcreJitStackMaxSizeScaleFactor                | Determines the maximum size of the Perl Compatible Regular Expressions (PCRE) just-in-time stack. The maximum stack size will be PcreJitStackMaxSizeScaleFactor * 1024 * 1024 bytes.  
**Default Value:** 32 |
| PatternMatchStackAllocator                   | Overrides the stack memory allocator for the pattern-match library when set to 1. The Perl Compatible Regular Expressions (PCRE) pattern-match library evaluates regular expressions. Restart the database for this parameter to take effect.  
**Default Value:** 1  
See Regular Expression Functions. |
| SegmentAutoProjection                        | Determines whether auto-projections are segmented by default. Set to 0 to disable.  
**Default Value:** 1 |
<p>| TerraceRoutingFactor                         | Specifies a value large enough that it cannot be enabled by default, even for the largest clusters. Use the Terrace Routing |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>equation to find the appropriate value for your cluster.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>1000.0</td>
</tr>
<tr>
<td>See Terrace Routing.</td>
<td></td>
</tr>
<tr>
<td>TransactionIsolationLevel</td>
<td>Changes the isolation level for the database. After modification, Vertica uses the new transaction level for every new session. Existing sessions and their transactions continue to use the original isolation level.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>See Change Transaction Isolation Levels.</td>
<td></td>
</tr>
<tr>
<td>TransactionMode</td>
<td>Specifies whether transactions are in read/write or read-only modes. Read/write is the default. Existing sessions and their transactions continue to use the original isolation level.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>READ WRITEs</td>
</tr>
<tr>
<td>UDxFencedBlockTimeout</td>
<td>Specifies the number of seconds to wait for output before aborting a UDx running in Fenced Mode. If the server aborts a UDx for this reason, it produces an error message similar to &quot;ERROR 3399: Failure in UDx RPC call: timed out in receiving a UDx message&quot;. If you see this error frequently, you can increase this limit. UDxs running in fenced mode do not run in the server process, so increasing this value does not impede server performance.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>60</td>
</tr>
<tr>
<td>UseLocalTzForParquetTimestampConversion</td>
<td>Specifies whether to do timezone conversion when reading Parquet files. Hive version 1.2.1 introduced an option to</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>localize timezones when writing Parquet files. Previously it wrote them in UTC and Vertica adjusted the value when reading the files. Set this parameter to 0 if Hive has already adjusted the timezones. Default Value: 1 (enabled)</td>
</tr>
</tbody>
</table>

**Tuple Mover Parameters**

These parameters control how the Tuple Mover operates.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivePartitionCount</td>
<td>Sets the number of active partitions. The active partitions are those most recently created. For example:</td>
</tr>
<tr>
<td></td>
<td>=&gt; ALTER DATABASE mydb SET ActivePartitionCount = 2;</td>
</tr>
<tr>
<td></td>
<td>For information about how the Tuple Mover treats active (and inactive) partitions during a mergeout operation, see Mergeout.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 1</td>
</tr>
<tr>
<td>CancelTMTimeout</td>
<td>When partition, copy table, and rebalance operations encounter a conflict with an internal Tuple Mover job, those operations attempt to cancel the conflicting Tuple Mover job. This parameter specifies the amount of time, in seconds, that the blocked operation waits for the Tuple Mover cancellation to take effect. If the operation is unable to cancel the Tuple Mover job within limit specified by this parameter, the operation displays an error and rolls back. Default Value: 300</td>
</tr>
<tr>
<td>EnableTMOnRecoveringNode</td>
<td>When enabled, allows Tuple Mover to perform moveout and mergeout activities on nodes with a node state of RECOVERING. Enabling Tuple Mover reduces the number</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>of ROS containers generated during recovery. Having fewer than 1024 ROS containers per projection allows Vertica to maintain optimal recovery performance. Default Value: 1</td>
</tr>
<tr>
<td>MaxMrgOutROSSizeMB</td>
<td>Specifies in MB the maximum size of ROS containers that are candidates for <code>mergeout</code> operations. The Tuple Mover avoids merging ROS containers that are larger than this setting. Default Value: -1 (no maximum limit)</td>
</tr>
<tr>
<td>MergeOutInterval</td>
<td>Specifies in seconds how long the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, consider a value less than the default. Default Value: 600</td>
</tr>
<tr>
<td>MoveOutInterval</td>
<td>Specifies in seconds how long Tuple Mover waits between checks for new data in the WOS to move to ROS. Default Value: 300</td>
</tr>
<tr>
<td>MoveOutMaxAgeTime</td>
<td>Specifies in seconds how long the Tuple Mover waits before it is forced to write the WOS to disk. Default Value: 1800</td>
</tr>
<tr>
<td>MoveOutSizePct</td>
<td>The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation. Default Value: 0</td>
</tr>
<tr>
<td>PurgeMergeoutPercent</td>
<td>Specifies as a percentage the threshold of deleted records in a ROS container that invokes an automatic <code>mergeout</code> operation, to purge those records. Vertica only counts the number of 'aged-out' delete vectors—that is, delete vectors that are as 'old' or older than the ancient history mark (AHM) epoch. This threshold applies to all ROS containers for non-partitioned tables. It also applies to ROS containers of all inactive partitions. In both cases, aged-out delete vectors</td>
</tr>
</tbody>
</table>
### Parameters

**Are permanently purged from the ROS container.**

**Note:** This configuration parameter only applies to automatic mergeout operations. It does not apply to manual mergeout operations that are invoked by calling meta-functions `DO_TM_TASK('mergeout')` and `PURGE`.

**Default Value:** 20 (percent)

### Projection Parameters

The following configuration parameters help you manage projections.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AnalyzeRowCountInterval</strong></td>
<td>Specifies how often Vertica checks the number of projection rows and whether the threshold set by <code>ARCCommitPercentage</code> has been crossed. For more information, see Collecting Statistics. <strong>Default Value:</strong> 60 seconds</td>
</tr>
<tr>
<td><strong>ARCCommitPercentage</strong></td>
<td>Sets the threshold percentage of WOS to ROS rows, which determines when to aggregate projection row counts and commit the result to the catalog. Vertica performs this action when the WOS to ROS percentage exceeds this setting. <strong>Default Value:</strong> 3 (percent)</td>
</tr>
<tr>
<td><strong>ContainersPerProjectionLimit</strong></td>
<td>Specifies how many ROS containers Vertica creates per projection before ROS pushback occurs. <strong>Default Value:</strong> 1024</td>
</tr>
</tbody>
</table>

**Caution:** Increasing this parameter's value can cause serious degradation of database performance. Vertica strongly recommends not to modify this parameter without first investigating the performance implications.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableGroupByProjections</td>
<td>When set to 1, you can create live aggregate projections. For more information, see <a href="#">Live Aggregate Projections</a>. Default Value: 1</td>
</tr>
<tr>
<td>EnableExprsInProjections</td>
<td>When set to 1, you can create projections that use expressions to calculate column values. For more information, see <a href="#">Aggregating Data Through Expressions</a>. Default Value: 1</td>
</tr>
<tr>
<td>EnableTopKProjections</td>
<td>When set to 1, you can create Top-K projections that let you retrieve Top-K data quickly. For more information, see <a href="#">Top-K Projections</a>. Default Value: 1</td>
</tr>
<tr>
<td>MaxAutoSegColumns</td>
<td>Specifies the number of columns (0–1024) to segment automatically when creating auto-projections from COPY and INSERT INTO statements. Set to 0 to use all columns in the hash segmentation expression. Default Value: 32</td>
</tr>
<tr>
<td>RebalanceQueryStorageContainers</td>
<td>By default, prior to performing a rebalance, Vertica performs a system table query to compute the size of all projections involved in the rebalance task. This query enables Vertica to optimize the rebalance to most efficiently utilize available disk space. This query can, however, significantly increase the time required to perform the rebalance. By disabling the system table query, you can reduce the time required to perform the rebalance. If your nodes are low on disk space, disabling the query increases the chance that a node runs out of disk space. In that situation, the rebalance fails.</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default Value: 1</td>
</tr>
<tr>
<td>SegmentAutoProjection</td>
<td>Determines whether auto-projections are segmented by default. Set to 0 to disable.</td>
</tr>
</tbody>
</table>

### Epoch Management Parameters

The following table describes the epoch management parameters for configuring Vertica.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdvanceAHMInterval</td>
<td>Determines how frequently (in seconds) Vertica checks the history retention status.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> AdvanceAHMInterval cannot be set to a value that is less than the EpochMapInterval.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 180 (3 minutes)</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> ALTER DATABASE mydb SET AdvanceAHMInterval = '3600';</td>
</tr>
<tr>
<td>AHMBackupManagement</td>
<td>Blocks the advancement of the Ancient History Mark (AHM). When this parameter is enabled, the AHM epoch cannot be later than the epoch of your latest full backup. If you advance the AHM to purge and delete data, do not enable this parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not enable this parameter before taking full backups, as it would prevent the AHM from advancing.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 0</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> ALTER DATABASE mydb SET AHMBackupManagement = '1';</td>
</tr>
<tr>
<td>EpochMapInterval</td>
<td>Determines the granularity of mapping between epochs and time available to historical queries. When a historical queries AT TIME T request is issued, Vertica maps it to an epoch within a granularity of EpochMapInterval seconds. It similarly affects the time reported for Last Good Epoch during Failure Recovery. Note that it does not affect internal precision of</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| epochs               | epochs themselves.                                                                                                                  |**Tip:** Decreasing this interval increases the number of epochs saved on disk. Therefore, consider reducing the HistoryRetentionTime parameter to limit the number of history epochs that Vertica retains.  
**Default Value:** 180 (3 minutes)  
**Example:**  
`ALTER DATABASE mydb SET EpochMapInterval = '300';` |
| HistoryRetentionTime | Determines how long deleted data is saved (in seconds) as an historical reference. When the specified time since the deletion has passed, you can purge the data. Use the -1 setting if you prefer to use HistoryRetentionEpochs to determine which deleted data can be purged.  
**Note:** The default setting of 0 effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM.  
**Tip:** If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window to remove loaded data. For example:  
`ALTER DATABASE mydb SET HistoryRetentionTime = 86400;`  
**Default Value:** 0 (Data saved when nodes are down.)  
**Example:**  
`ALTER DATABASE mydb SET HistoryRetentionTime = '240';` |
| HistoryRetentionEpochs | Specifies the number of historical epochs to save, and therefore, the amount of deleted data. Unless you have a reason to limit the number of epochs, Vertica recommends that you specify the time over which deleted data is saved.  
If you specify both History parameters, HistoryRetentionTime takes precedence. Setting both parameters to -1, preserves all historical data. |
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HistoryRetentionEpochs</td>
<td>See Setting a Purge Policy. Default Value: -1 (Disabled) Example: ALTER DATABASE mydb SET HistoryRetentionEpochs = '40';</td>
</tr>
</tbody>
</table>

### Monitoring Parameters

The following table describes the monitoring parameters for configuring Vertica.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SnmpTrapDestinationsList</td>
<td>Defines where Vertica sends traps for SNMP. See Configuring Reporting for SNMP. Default Value: none Example: ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';</td>
</tr>
<tr>
<td>SnmpTrapsEnabled</td>
<td>Enables event trapping for SNMP. See Configuring Reporting for SNMP. Default Value: 0 Example: ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;</td>
</tr>
<tr>
<td>SyslogEnabled</td>
<td>Enables event trapping for syslog. See Configuring Reporting for Syslog.</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value: 0</td>
<td>Example: ALTER DATABASE mydb SET SyslogEnabled = 1;</td>
</tr>
<tr>
<td>SyslogEvents</td>
<td>Defines events that generate a syslog entry. See Configuring Reporting for Syslog.</td>
</tr>
<tr>
<td>Default Value: none</td>
<td>Example: ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';</td>
</tr>
<tr>
<td>SyslogFacility</td>
<td>Defines which SyslogFacility Vertica uses. See Configuring Reporting for Syslog.</td>
</tr>
<tr>
<td>Default Value: user</td>
<td>Example: ALTER DATABASE mydb SET SyslogFacility = 'ftp';</td>
</tr>
</tbody>
</table>

### Profiling Parameters

The following table describes the profiling parameters for configuring Vertica. See Profiling Database Performance for more information on profiling queries.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalEEProfiling</td>
<td>Enables profiling for query execution runs in all sessions on all nodes.</td>
</tr>
<tr>
<td>Default Value: 0</td>
<td>Example: ALTER DATABASE mydb SET GlobalEEProfiling = 1;</td>
</tr>
<tr>
<td>GlobalQueryProfiling</td>
<td>Enables query profiling for all sessions on all nodes.</td>
</tr>
<tr>
<td>Default Value: 0</td>
<td>Example: ALTER DATABASE mydb SET GlobalQueryProfiling = 1;</td>
</tr>
<tr>
<td>GlobalSessionProfiling</td>
<td>Enables session profiling for all sessions on all nodes.</td>
</tr>
</tbody>
</table>
### Security Parameters

Use these client authentication configuration parameters and general security parameters to configure security.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultIdleSessionTimeout</td>
<td>Indicates a default session timeout value for all users where IDLESESSIONTIMEOUT is not set.</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td>ALTER DATABASE mydb SET GlobalSessionProfiling = 1;</td>
</tr>
<tr>
<td>DoUserSpecificFilteringInSysTables</td>
<td>Specifies whether a non-superuser can view details of another user, one of the following:</td>
</tr>
<tr>
<td></td>
<td>- 0: Users can view details of other.</td>
</tr>
<tr>
<td></td>
<td>- 1: Users can only view details about themselves.</td>
</tr>
<tr>
<td>EnableAllRolesOnLogin</td>
<td>Automatically enables all roles granted to a user on login, one of the following:</td>
</tr>
<tr>
<td></td>
<td>- 0 (default): Do not automatically enable roles</td>
</tr>
<tr>
<td></td>
<td>- 1: Automatically enable roles. With this setting, users do not need to run SET ROLE</td>
</tr>
<tr>
<td>EnabledCipherSuites</td>
<td>Specifies which SSL cipher suites to use for secure client-server communication.</td>
</tr>
</tbody>
</table>
|                               | **Default Value:**
<p>|                               | ALL:!ADH:!LOW:!EXP:!MD5:!RC4:@STRENGTH                                                                                                      |
|                               | This setting excludes weaker cipher suites.                                                                                                  |
|                               | For a complete mapping of cipher suite names from JSSE to OpenSSL, see openssl.org.                                                            |</p>
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableSSL</td>
<td>Enables SSL for the server, set to one of the following:</td>
</tr>
<tr>
<td></td>
<td>• 0 (default): Disable</td>
</tr>
<tr>
<td></td>
<td>• 1: Enable</td>
</tr>
<tr>
<td></td>
<td>For example:                                                                elts DATABASE DEFAULT SET EnableSSL = '1';</td>
</tr>
<tr>
<td></td>
<td>For details, see Implementing SSL.</td>
</tr>
<tr>
<td>GlobalHeirUserName</td>
<td>A string that specifies which user inherits objects after their owners are dropped. This setting ensures preservation of data otherwise lost.</td>
</tr>
<tr>
<td></td>
<td>Set this parameter to one of the following string values:</td>
</tr>
<tr>
<td></td>
<td>• Empty string: Objects of dropped users are removed from the database.</td>
</tr>
<tr>
<td></td>
<td>• <strong>username</strong>: Reassigns objects of dropped users to <strong>username</strong>. If <strong>username</strong> does not exist, Vertica creates that user and sets GlobalHeirUserName to it.</td>
</tr>
<tr>
<td></td>
<td>• &lt;auto&gt; (default): Reassigns objects of dropped LDAP users to user dbadmin.</td>
</tr>
<tr>
<td></td>
<td>Note: Be sure to include the angle brackets &lt; &gt;.</td>
</tr>
<tr>
<td></td>
<td>See below for an example.</td>
</tr>
<tr>
<td>RequireFIPS</td>
<td>Specifies whether the FIPS mode is enabled or disabled. You cannot modify this parameter. The value of this parameter matches the contents of the file crypto.fips_enabled. On startup, Vertica automatically sets this parameter to one of the following.</td>
</tr>
<tr>
<td></td>
<td>0 - Disabled FIPS</td>
</tr>
<tr>
<td></td>
<td>1 - Enabled FIPS</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SecurityAlgorithm</td>
<td>Sets the algorithm for the function that hash authentication uses, one of the following: &lt;br&gt;• MD5 &lt;br&gt;• SHA-512 &lt;br&gt;For example: ALTER DATABASE DEFAULT SET SecurityAlgorithm = 'SHA512'; Default Value: 'NONE'</td>
</tr>
<tr>
<td>SSLCA</td>
<td>Sets the SSL certificate authority. Include the contents of the certificate authority root.crt file, but exclude the file name. For example: ALTER DATABASE DEFAULT SET SSLCA = 'contents of root.crt file';</td>
</tr>
<tr>
<td>SSLCertificate</td>
<td>Sets the SSL certificate. Include the contents of the server.crt file, but exclude the file name. If your SSL certificate is a certificate chain, set this parameter to the contents of from the top-most certificate of the certificate chain. For example: ALTER DATABASE DEFAULT SET SSLCertificate = 'contents of server.crt file';</td>
</tr>
<tr>
<td>SSLPrivateKey</td>
<td>The server's private key, visible only to dbadmin users. This parameter is set to the contents of the server.key file; it excludes the file name. For example: ALTER DATABASE DEFAULT SET SSLPrivateKey = 'contents of server.key file';</td>
</tr>
</tbody>
</table>

**Examples**

Set security parameter value GlobalHeirUserName:
See Also

- **Kerberos Authentication Parameters**
- **Configuring SSL**

**Database Designer Parameters**

The following table describes the parameters for configuring the Vertica Database Designer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBDCorrelationSampleRowCount</td>
<td>Minimum number of table rows at which</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Database Designer discovers and records</td>
<td>correlated columns.</td>
</tr>
<tr>
<td>DBDLogInternalDesignProcess</td>
<td>Enables or disables Database Designer logging.</td>
</tr>
<tr>
<td>DBDUseOnlyDesignerResourcePool</td>
<td>Enables use of the DBD pool by the Vertica Database Designer.</td>
</tr>
<tr>
<td></td>
<td>When set to false, design processing is mostly contained by the user's</td>
</tr>
<tr>
<td></td>
<td>resource pool, but might spill over into some system resource pools for</td>
</tr>
<tr>
<td></td>
<td>less-intensive tasks</td>
</tr>
</tbody>
</table>

### Internationalization Parameters

The following table describes the internationalization parameters for configuring Vertica.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultIntervalStyle</td>
<td>Sets the default interval style to use. If set to 0 (default), the interval is in PLAIN style (the SQL standard), no interval units on output. If set to 1, the interval is in UNITS on output. This parameter does not take effect until the database is restarted.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 0</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> ALTER DATABASE mydb SET DefaultIntervalStyle = 1;</td>
</tr>
<tr>
<td>DefaultSessionLocale</td>
<td>Sets the default session startup locale for the database. This parameter does not take effect until the database is restarted.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> en_US@collation=binary</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> ALTER DATABASE mydb SET DefaultSessionLocale = 'en_GB';</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| EscapeStringWarning        | Issues a warning when back slashes are used in a string literal. This is provided to help locate back slashes that are being treated as escape characters so they can be fixed to follow the Standard conforming string syntax instead.  

**Default Value:** 1  

**Example:**  
ALTER DATABASE mydb SET EscapeStringWarning = '1'; |
| StandardConformingStrings  | Determines whether ordinary string literals ('...') treat backslashes (\) as string literals or escape characters. When set to '1', backslashes are treated as string literals, when set to '0', back slashes are treated as escape characters.  

**Tip:** To treat backslashes as escape characters, use the Extended string syntax:  
(E'...');  

**Default Value:** 1  

**Example:**  
ALTER DATABASE mydb SET StandardConformingStrings = '0'; |

---

### Data Collector Parameters

The following table lists the Data Collector parameter for configuring Vertica.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| EnableDataCollector     | Enables and disables the Data Collector, which is the Workload Analyzer's internal diagnostics utility. Affects all sessions on all nodes. Use 0 to turn off data collection.  

**Default value:** 1 (Enabled)  

**Example:**  
ALTER DATABASE mydb SET EnableDataCollector = 0; |
Text Search Parameters

You can configure Vertica for text search using these parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextIndexMaxTokenLength</td>
<td>Controls the maximum size of a token in a text index. If the parameter is set to a value greater than 65000 characters, then the tokenizer truncates the token at 65000 characters.</td>
</tr>
<tr>
<td></td>
<td>You should avoid setting this parameter near 65000 (the maximum value). Doing so can result in a significant decrease in performance. For optimal performance, the parameter should be set to the maximum token value of your tokenizer.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 128 characters</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong> ALTER DATABASE database_name SET PARAMETER TextIndexMaxTokenLength=760;</td>
</tr>
</tbody>
</table>

Kerberos Configuration Parameters

The following parameters let you configure the Vertica principal for Kerberos authentication and specify the location of the Kerberos keytab file.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KerberosServiceName</td>
<td>Provides the service name portion of the Vertica Kerberos principal. By default, this parameter is 'vertica'. For example: <em>vertica</em>/host@EXAMPLE.COM.</td>
</tr>
</tbody>
</table>
| KerberosHostname       | [Optional] Provides the instance or host name portion of the Vertica Kerberos principal. For example: *vertica/host@EXAMPLE.COM*  
If you omit the optional KerberosHostname parameter, Vertica uses the return value from the gethostname() function. Assuming each cluster node has a different host name, those nodes will each have a different principal, which you must manage in that node's keytab file. |
| KerberosRealm          | Provides the realm portion of the Vertica Kerberos principal. A realm is the authentication administrative domain and is usually formed in uppercase letters; for example: *vertica/host@EXAMPLE.COM*. |
| KerberosKeytabFile     | Provides the location of the keytab file that contains credentials for the Vertica Kerberos principal. By default, this file is located in /etc. For example: *KerberosKeytabFile=/etc/krb5.keytab*.  
**Notes:**  
- The principal must take the form  
  KerberosServiceName/KerberosHostName@KerberosRealm  
- The keytab file must be readable by the file owner who is running the process (typically the Linux dbadmin user assigned file permissions 0600). |
| KerberosTicketDuration | Determines the lifetime of the ticket retrieved from performing a kinit. The default is 0 (zero) which disables this parameter.  
If you do not enter a value here, the lifetime is determined by the default Kerberos configuration. |
Eon Mode Beta Parameters

The following parameters configure how the database operates when running in Eon Mode Beta. See Using Eon Mode Beta.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| DelayForDeletes      | Specifies the time interval to wait before deleting a file from communal storage. When you delete a file, Vertica first immediately deletes it from the depot. After the specified time interval, the delete also occurs communal storage.  
  **Default Value:** 2 hours |
| TxnLogSyncInterval   | Sets a time interval for which the metadata is asynchronously pushed to shared storage.                                                        |
| UseDepotForReads     | Indicates whether Vertica accesses the depot to answer queries, or accesses only communal storage. Depot reads are enabled to improve query performance and provide database K-safety in Eon Mode Beta. To see how your system runs without the depot, disable this parameter.  
  **Default Value:** 1 (enabled)  
  **Valid Values:**  
  - 1: Queries can read from the depot. The depot fetches data from communal storage to answer queries.  
  - 0: Reading from the depot is disabled. Queries will only read from communal storage. |

Apache Hadoop Parameters

The following table describes the general parameters for configuring integration with Apache Hadoop. See Integrating with Apache Hadoop for more information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HadoopConfDir</td>
<td>A directory path containing the XML configuration files copied from Hadoop. The same path must be valid on every Vertica node. You can</td>
</tr>
</tbody>
</table>
use the VERIFY_HADOOP_CONF_DIR meta-function to test that the value is set correctly. Setting this parameter is required to read data from HDFS.

When you set this parameter, any previously-cached configuration information is flushed.

**Default Value:** obtained from environment if possible

**Requires Restart:** No

**Example:**
```
ALTER DATABASE mydb SET HadoopConfDir = '/hadoop/hcat/conf';
```

The following table describes the parameters for configuring the HCatalog Connector. See Using the HCatalog Connector in Integrating with Apache Hadoop for more information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableHCatImpersonation</td>
<td>Whether the HCatalog Connector uses (impersonates) the current Vertica user when accessing Hive. If impersonation is enabled, the HCatalog Connector uses the Kerberos credentials of the logged-in Vertica user to access Hive data. Disable impersonation if you are using an authorization service to manage access without also granting users access to the underlying files. For more information, see Configuring Kerberos in Integrating with Apache Hadoop.</td>
</tr>
<tr>
<td><strong>Default Value:</strong> 1 (enabled)</td>
<td></td>
</tr>
<tr>
<td><strong>Requires Restart:</strong> No</td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>ALTER DATABASE mydb SET EnableHCatImpersonation = 0;</code></td>
</tr>
<tr>
<td>HCatalogConnectorUseHiveServer2</td>
<td>When enabled, Vertica internally uses HiveServer2 instead of WebHCat to get metadata from Hive.</td>
</tr>
<tr>
<td><strong>Default Value:</strong> 1 (enabled)</td>
<td></td>
</tr>
<tr>
<td><strong>Requires Restart:</strong> No</td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>ALTER DATABASE mydb SET HCatalogConnectorUseHiveServer2 = 0;</code></td>
</tr>
<tr>
<td>HCatalogConnectorUseLibHDFSPP</td>
<td>Whether the HCatalog Connector should use the hdfs scheme instead of webhdfs to read native</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HCatalogConnectorUseLibHDFSPP</td>
<td>Using the hdfs scheme requires additional configuration but can have better performance.</td>
</tr>
<tr>
<td>HCatConnectionTimeout</td>
<td>The number of seconds the HCatalog Connector waits for a successful connection to the HiveServer2 (or WebHCat) server before returning a timeout error.</td>
</tr>
<tr>
<td>HCatSlowTransferLimit</td>
<td>The lowest transfer speed (in bytes per second) that the HCatalog Connector allows when retrieving data from the HiveServer2 (or WebHCat) server. In some cases, the data transfer rate from the server to Vertica is below this threshold. In such cases, after the number of seconds specified in the HCatSlowTransferTime parameter pass, the HCatalog Connector cancels the query and closes the connection.</td>
</tr>
<tr>
<td>HCatSlowTransferTime</td>
<td>The number of seconds the HCatalog Connector waits before testing whether the data transfer from the server is too slow. See the HCatSlowTransferLimit parameter.</td>
</tr>
</tbody>
</table>
Note: You can override the HCatalog configuration parameters when creating an HCatalog schema. See CREATE HCATALOG SCHEMA in the SQL Reference Manual for an explanation.

AWS Parameters

Use the following parameters to configure reading from S3 using COPY FROM. For more information about reading data from S3, see Specifying COPY FROM Options.

For the parameters to control the AWS Library (UDSource), see Configuring the Vertica Library for Amazon Web Services.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWSCAFile</td>
<td>The file name of the SSL server certificate bundle to use. For SUSE Linux you must set a value.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> system-dependent</td>
</tr>
<tr>
<td>AWSCAPath</td>
<td>The path Vertica uses to look up SSL server certificates. For SUSE Linux you must set a value.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> system-dependent</td>
</tr>
<tr>
<td>AWSEnableHttps</td>
<td>Whether to use the HTTPS protocol when connecting to S3.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 1 (enabled)</td>
</tr>
<tr>
<td>AWSEndpoint</td>
<td>The connection endpoint address.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> &quot; (use Amazon S3 servers)</td>
</tr>
<tr>
<td>AWSRegion</td>
<td>The region containing the S3 bucket from which to read files. AWSRegion can only be configured with one region at a time. If you need to access buckets in multiple regions, change the parameter each time you change regions. If you do not set the correct region, you might experience a delay before queries fail because Vertica retries several times before giving up.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> us-east-1</td>
</tr>
<tr>
<td></td>
<td>You can find more information about AWS regions in the Amazon Documentation.</td>
</tr>
</tbody>
</table>
User-Defined Session Parameters

Use the following Vertica use-defined session parameters to configure Kafka SSL when not using a scheduler.

The kafka_ parameters configure SSL authentication for Kafka. Refer to Using SSL with Kafka for more information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka_SSL_CA</td>
<td>The contents of the certificate authority certificate.</td>
</tr>
<tr>
<td>Default Value: none</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>=&gt; ALTER SESSION SET UDPARAMETER kafka_SSL_CA='MIIBOQIBAAJBAIOL';</td>
</tr>
<tr>
<td>kafka_SSL_Certificate</td>
<td>The contents of the SSL certificate.</td>
</tr>
<tr>
<td>Default Value: none</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>=&gt; ALTER SESSION SET UDPARAMETER kafka_SSL_Certificate='XrM07O4dV/nJ5g';</td>
</tr>
<tr>
<td>kafka_SSL_PrivateKey_secret</td>
<td>The private key used to encrypt the session. Vertica does not log this information.</td>
</tr>
<tr>
<td>Default Value: none</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>=&gt; ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKey_secret='A60iThKtezaCk7F';</td>
</tr>
<tr>
<td>kafka_SSL_PrivateKeyPassword_secret</td>
<td>The password used to create the private key. Vertica does not log this information.</td>
</tr>
<tr>
<td>Default Value: none</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKeyPassword_secret='secret';</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka_Enable_SSL</td>
<td>Enables SSL authentication for Vertica-Kafka integration.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 0</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET kafka_Enable_SSL=1;</code></td>
</tr>
<tr>
<td>MaxSessionUDParameterSize</td>
<td>Sets the maximum length of a value in a user-defined session parameter.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 1000</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET MaxSessionUDParameterSize = 2000</code></td>
</tr>
</tbody>
</table>

### Related Topics

- User-Defined Session Parameters

### Constraint Enforcement Parameters

The following configuration parameters enforce constraints.

Use the `ALTER DATABASE` statement to set these parameters. You do not need to restart your database after setting them.

- The parameter settings apply for any check constraint, and any primary or unique key constraint that you have not explicitly enabled or disabled within a `CREATE TABLE` or `ALTER TABLE` statement.

- Any new check constraint, and any primary or unique key constraint that you create or alter is set according to the value of the corresponding parameter unless you specifically enabled or disabled the constraint.

**Important:** Setting a constraint as enabled or disabled when you create or alter it using `CREATE TABLE` or `ALTER TABLE` overrides the parameter setting.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
</table>
| EnableNewCheckConstraintsByDefault | Set to 1 (the default) to automatically enable newly created check constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced. **Default Value: 1 (Enabled)**  
**Example:**  
```
ALTER DATABASE mydb SET EnableNewCheckConstraintsByDefault = 0;
``` |
| EnableNewPrimaryKeysByDefault | Set to 1 to automatically enable newly created primary key constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced. **Default Value: 0 (Disabled)**  
**Example:**  
```
ALTER DATABASE mydb SET EnableNewPrimaryKeysByDefault = 1;
``` |
| EnableNewUniqueKeysByDefault | Set to 1 to automatically enable newly created unique constraints that you specified through CREATE TABLE or ALTER TABLE statements. However, if you have explicitly disabled a constraint when you created or altered it, it is not enforced. **Default Value: 0 (Disabled)**  
**Example:**  
```
ALTER DATABASE mydb SET EnableNewUniqueKeysByDefault = 1;
``` |

**Note:** Vertica recommends enabling primary key enforcement if you have enabled unique key enforcement.
## Numeric Precision Parameters

The following configuration parameters let you configure numeric precision for numeric data types. For more about using these parameters, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowNumericOverflow</td>
<td>When set to 1 (True), allows silent numeric overflow. When true, Vertica does not implicitly extend precision of numeric data types. <strong>Default Value: 1 (True)</strong> When set to 0 (False), Vertica produces an overflow error, if a result exceeds the precision set by NumericSumExtraPrecisionDigits. Vertica ignores the value of NumericSumExtraPrecisionDigits when AllowNumericOverflow is true. <strong>Example:</strong> ALTER DATABASE mydb SET PARAMETER AllowNumericOverflow=0;</td>
</tr>
<tr>
<td>NumericSumExtraPrecisionDigits</td>
<td>Vertica produces an overflow error, if a result exceeds the specified precision, By default this specified precision is six places beyond the DDL-specified precision. <strong>Default Value: 6</strong> This parameter setting only applies if AllowNumericOverflow is set to 0 (False). <strong>Possible Values: 0 to 20.</strong> <strong>Example:</strong> ALTER DATABASE mydb SET PARAMETER NumericSumExtraPrecisionDigits=8</td>
</tr>
</tbody>
</table>
Vertica Library for Amazon Web Services Parameters

Use these parameters to configure the VerticaLibrary for Amazon Web Services (AWS). All parameters listed are case-sensitive.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws_id</td>
<td>Your AWS access key ID.</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET UDPARAMETER FOR awslib aws_id='aws-id';</code></td>
</tr>
<tr>
<td>aws_secret</td>
<td>Your AWS secret access key.</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET UDPARAMETER FOR awslib aws_secret='aws-key';</code></td>
</tr>
<tr>
<td>aws_region</td>
<td>The region containing your S3 bucket. aws_region can only be configured with</td>
</tr>
<tr>
<td></td>
<td>one region at a time. If you need to access buckets in multiple regions, you</td>
</tr>
<tr>
<td></td>
<td>must re-set the parameter each time you change regions.</td>
</tr>
<tr>
<td></td>
<td>This parameter is different from AWSRegion, which is used with COPY FROM as</td>
</tr>
<tr>
<td></td>
<td>described in Specifying COPY FROM Options.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> us-east-1</td>
</tr>
<tr>
<td></td>
<td>You can find more information about AWS regions in the Amazon Documentation.</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET UDPARAMETER FOR awslib aws_region='region-id';</code></td>
</tr>
<tr>
<td>aws_ca_path</td>
<td>The path which Vertica will use when looking for SSL server certificates.</td>
</tr>
<tr>
<td></td>
<td>For SUSE Linux you must set a value.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> system dependent</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td><code>=&gt; ALTER SESSION SET UDPARAMETER FOR awslib aws_ca_path='/home/user/ssl_folder';</code></td>
</tr>
<tr>
<td>aws_ca_bundle</td>
<td>The path which Vertica will use when looking for a SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>bundle. For SUSE Linux you must set a value.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Default value:</strong></td>
<td>system dependent</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>ALTER SESSION SET UDPARAMETER FOR awslib aws_ca_bundle='/home/user/ssl_folder/ca_bundle';</code></td>
</tr>
<tr>
<td>aws_proxy</td>
<td>A string value which allows you to set an HTTP/HTTPS proxy for the AWS library.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>ALTER SESSION SET UDPARAMETER FOR awslib aws_proxy='192.168.1.1:8080';</code></td>
</tr>
<tr>
<td>aws_verbose</td>
<td>When enabled, logs libcurl debug messages to /opt/vertica/packages/AWS/logs.</td>
</tr>
<tr>
<td><strong>Default value:</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><code>ALTER SESSION SET UDPARAMETER FOR awslib aws_verbose=1;</code></td>
</tr>
<tr>
<td>aws_max_send_speed</td>
<td>The maximum transfer speed when sending data to AWS S3 in bytes per second.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>The following example sets a maximum send speed of 1KB/S:</td>
</tr>
<tr>
<td></td>
<td><code>ALTER SESSION SET UDPARAMETER FOR awslib aws_max_send_speed=1024;</code></td>
</tr>
<tr>
<td>aws_max_recv_speed</td>
<td>The maximum transfer speed when receiving data to AWS S3 in bytes per second.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>The following example sets a maximum receive speed of 100KB/S:</td>
</tr>
<tr>
<td></td>
<td><code>ALTER SESSION SET UDPARAMETER FOR awslib aws_max_recv_speed=102400;</code></td>
</tr>
<tr>
<td>aws_endpoint</td>
<td>The AWS endpoint to use when interpreting S3 URLs, in the form http(s)://host:port.</td>
</tr>
<tr>
<td><strong>Default value:</strong></td>
<td><a href="https://s3.amazonaws.com">https://s3.amazonaws.com</a></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>= ALTER SESSION SET UDFPARAMETER FOR awslib aws_endpoint='<a href="http://localhost:8080">http://localhost:8080</a>';</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- [AWS Library](#)
- **Configuring Vertica AWS Library**
- [Export AWS Library](#)
- [Import AWS Library](#)
Designing a Logical Schema

Designing a logical schema for a Vertica database is the same as designing for any other SQL database. A logical schema consists of objects such as schemas, tables, views and referential Integrity constraints that are visible to SQL users. Vertica supports any relational schema design that you choose.
Using Multiple Schemas

Using a single schema is effective if there is only one database user or if a few users cooperate in sharing the database. In many cases, however, it makes sense to use additional schemas to allow users and their applications to create and access tables in separate namespaces. For example, using additional schemas allows:

- Many users to access the database without interfering with one another.

  Individual schemas can be configured to grant specific users access to the schema and its tables while restricting others.

- Third-party applications to create tables that have the same name in different schemas, preventing table collisions.

Unlike other RDBMS, a schema in a Vertica database is not a collection of objects bound to one user.

Multiple Schema Examples

This section provides examples of when and how you might want to use multiple schemas to separate database users. These examples fall into two categories: using multiple private schemas and using a combination of private schemas (i.e. schemas limited to a single user) and shared schemas (i.e. schemas shared across multiple users).

Using Multiple Private Schemas

Using multiple private schemas is an effective way of separating database users from one another when sensitive information is involved. Typically a user is granted access to only one schema and its contents, thus providing database security at the schema level. Database users can be running different applications, multiple copies of the same application, or even multiple instances of the same application. This enables you to consolidate applications on one database to reduce management overhead and use resources more effectively. The following examples highlight using multiple private schemas.

Using multiple schemas to separate users and their unique applications

In this example, both database users work for the same company. One user (HRUser) uses a Human Resource (HR) application with access to sensitive personal data, such as salaries, while another user (MedUser) accesses information regarding company healthcare costs through a
healthcare management application. HRUser should not be able to access company healthcare cost information and MedUser should not be able to view personal employee data.

To grant these users access to data they need while restricting them from data they should not see, two schemas are created with appropriate user access, as follows:

- **HRSchema**—A schema owned by HRUser that is accessed by the HR application.
- **HealthSchema**—A schema owned by MedUser that is accessed by the healthcare management application.

**Using multiple schemas to support multitenancy**

This example is similar to the last example in that access to sensitive data is limited by separating users into different schemas. In this case, however, each user is using a virtual instance of the same application.

An example of this is a retail marketing analytics company that provides data and software as a service (SaaS) to large retailers to help them determine which promotional methods they use are most effective at driving customer sales.

In this example, each database user equates to a retailer, and each user only has access to its own schema. The retail marketing analytics company provides a virtual instance of the same application to each retail customer, and each instance points to the user’s specific schema in which to create and update tables. The tables in these schemas use the same names because they are created by instances of the same application, but they do not conflict because they are in separate schemas.

Example of schemas in this database could be:

- **MartSchema**—A schema owned by MartUser, a large department store chain.
- **PharmSchema**—A schema owned by PharmUser, a large drug store chain.

**Using multiple schemas to migrate to a newer version of an application**

Using multiple schemas is an effective way of migrating to a new version of a software application. In this case, a new schema is created to support the new version of the software, and the old schema is kept as long as necessary to support the original version of the software. This is called a “rolling application upgrade.”

For example, a company might use a HR application to store employee data. The following schemas could be used for the original and updated versions of the software:
HRSchema—A schema owned by HRUser, the schema user for the original HR application.

V2HRSchema—A schema owned by V2HRUser, the schema user for the new version of the HR application.

Combining Private and Shared Schemas

The previous examples illustrate cases in which all schemas in the database are private and no information is shared between users. However, users might want to share common data. In the retail case, for example, MartUser and PharmUser might want to compare their per store sales of a particular product against the industry per store sales average. Since this information is an industry average and is not specific to any retail chain, it can be placed in a schema on which both users are granted USAGE privileges. (For more information about schema privileges, see Schema Privileges.)

Example of schemas in this database might be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.
- IndustrySchema—A schema owned by DBUser (from the retail marketing analytics company) on which both MartUser and PharmUser have USAGE privileges. It is unlikely that retailers would be given any privileges beyond USAGE on the schema and SELECT on one or
more of its tables.

Creating Schemas

You can create as many schemas as necessary for your database. For example, you could create a schema for each database user. However, schemas and users are not synonymous as they are in Oracle.

By default, only a superuser can create a schema or give a user the right to create a schema. (See GRANT (Database) in the SQL Reference Manual.)

To create a schema use the CREATE SCHEMA statement, as described in the SQL Reference Manual.
Specifying Objects in Multiple Schemas

Once you create two or more schemas, each SQL statement or function must identify the schema associated with the object you are referencing. You can specify an object within multiple schemas by:

- Qualifying the object name by using the schema name and object name separated by a dot. For example, to specify MyTable, located in Schema1, qualify the name as Schema1.MyTable.

- Using a search path that includes the desired schemas when a referenced object is unqualified. By Setting Search Paths, Vertica will automatically search the specified schemas to find the object.

Setting Search Paths

Each user session has a search path of schemas. Vertica uses this search path to find tables and user-defined functions (UDFs) that are unqualified by their schema name. A session search path is initially set from the user's profile. You can change the session's search path at any time by calling SET SEARCH_PATH. This search path remains in effect until the next SET SEARCH_PATH statement, or the session ends.

Viewing the Current Search Path

SHOW SEARCH_PATH returns the session's current search path. For example:

```
=> SHOW SEARCH_PATH;
   name     | setting
------------+------------------------
 search_path | "$user", public, v_catalog, v_monitor, v_internal
```

Schemas are listed in descending order of precedence. The first schema has the highest precedence in the search order. If this schema exists, it is also defined as the current schema, which is used for tables that are created with unqualified names. You can identify the current schema by calling the function CURRENT_SCHEMA:

```
=> SELECT CURRENT_SCHEMA;
current_schema
------------------
public
(1 row)
```
Setting the User Search Path

A session search path is initially set from the user's profile. If the search path in a user profile is not set by `CREATE USER` or `ALTER USER`, it is set to the database default:

```
=> CREATE USER agent007;
CREATE USER
=> \c - agent007
You are now connected as user "agent007".
=> SHOW SEARCH_PATH;  setting
--------------------------
search_path | "$user", public, v_catalog, v_monitor, v_internal
```

$\textit{user}$ resolves to the session user name—in this case, \textit{agent007}—and has the highest precedence. If a schema \textit{agent007}, exists, Vertica begins searches for unqualified tables in that schema. Also, calls to `CURRENT_SCHEMA` return this schema. Otherwise, Vertica uses public as the current schema and begins searches in it.

Use `ALTER USER` to modify an existing user's search path. These changes overwrite all non-system schemas in the search path, including $\textit{USER}$. System schemas are untouched. Changes to a user's search path take effect only when the user starts a new session; current sessions are unaffected.

**Important:** After modifying the user's search path, verify that the user has access privileges to all schemas that are on the updated search path.

For example, the following statements modify \textit{agent007}'s search path, and grant access privileges to schemas and tables that are on the new search path:

```
=> ALTER USER agent007 SEARCH_PATH store, public;
ALTER USER
=> GRANT ALL ON SCHEMA store, public TO agent007;
GRANT PRIVILEGE
=> GRANT SELECT ON ALL TABLES IN SCHEMA store, public TO agent007;
GRANT PRIVILEGE
=> \c - agent007
You are now connected as user "agent007".
=> SHOW SEARCH_PATH;  setting
--------------------------
search_path | store, public, v_catalog, v_monitor, v_internal
(1 row)
```

To verify a user's search path, query the system table `USERS`:

```
=> SELECT search_path FROM USERS WHERE user_name='agent007';
search_path
--------------------------
```
To revert a user's search path to the database default settings, call `ALTER USER` and set the search path to DEFAULT. For example:

```
=> ALTER USER agent007 SEARCH_PATH DEFAULT;
=> SELECT search_path FROM USERS WHERE user_name='agent007';
```

```
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

Ignored Search Path Schemas

Vertica only searches among existing schemas to which the current user has access privileges. If a schema in the search path does not exist or the user lacks access privileges to it, Vertica silently excludes it from the search. For example, if `agent007` lacks SELECT privileges to schema `public`, Vertica silently skips this schema. Vertica returns with an error only if it cannot find the table anywhere on the search path.

Setting Session Search Path

Vertica initially sets a session's search path from the user's profile. You can change the current session's search path with `SET SEARCH_PATH`. You can use `SET SEARCH_PATH` in two ways:

- Explicitly set the session search path to one or more schemas. For example:

```
=> \c - agent007
You are now connected as user "agent007".
dbadmin=> SHOW SEARCH_PATH;

name | setting
---------------------------
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)

=> SET SEARCH_PATH TO store, public;
SET
=> SHOW SEARCH_PATH;

name | setting
---------------------------
search_path | store, public, v_catalog, v_monitor, v_internal
(1 row)
```
Set the session search path to the database default:

```sql
=> SET SEARCH_PATH TO DEFAULT;
SET
=> SHOW SEARCH_PATH;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>search_path</td>
<td>&quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
</tbody>
</table>

(1 row)

SET SEARCH_PATH overwrites all non-system schemas in the search path, including $USER. System schemas are untouched.

### Creating Objects That Span Multiple Schemas

Vertica supports views that reference tables across multiple schemas. For example, a user might need to compare employee salaries to industry averages. In this case, the application queries two schemas:

- **Shared schema** IndustrySchema for salary averages
- **Private schema** HRSchema for company-specific salary information

#### Best Practice: When creating objects that span schemas, use qualified table names. This naming convention avoids confusion if the query path or table structure within the schemas changes at a later date.

### Tables in Schemas

In Vertica you can create persistent and temporary tables, through `CREATE TABLE` and `CREATE TEMPORARY TABLE`, respectively.
For detailed information on both types, see Creating Tables and Creating Temporary Tables.

Persistent Tables

`CREATE TABLE` creates a table in the Vertica logical schema. For example:

```sql
CREATE TABLE vendor_dimension (    vendor_key INTEGER NOT NULL PRIMARY KEY,    vendor_name VARCHAR(64),    vendor_address VARCHAR(64),    vendor_city VARCHAR(64),    vendor_state CHAR(2),    vendor_region VARCHAR(32),    deal_size INTEGER,    last_deal_update DATE    );
```

For detailed information, see Creating Tables.

Temporary Tables

`CREATE TEMPORARY TABLE` creates a table whose data persists only during the current session. Temporary table data is never visible to other sessions.

Temporary tables can be used to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated—for example, the tool first gets a result set, then queries the result set, and so on.

`CREATE TEMPORARY TABLE` can create tables at two scopes, global and local, through the keywords GLOBAL and LOCAL, respectively:

- **GLOBAL** (default): The table definition is visible to all sessions. However, table data is session-scoped.
- **LOCAL**: The table definition is visible only to the session in which it is created. When the session ends, Vertica automatically drops the table.

For detailed information, see Creating Temporary Tables.
Creating a Database Design

A design is a physical storage plan that optimizes query performance. Data in Vertica is physically stored in projections. When you initially load data into a table using INSERT, COPY (or COPY LOCAL), Vertica creates a default superprojection for the table. This superprojection ensures that all of the data is available for queries. However, these superprojections might not optimize database performance, resulting in slow query performance and low data compression.

To improve performance, create a design for your Vertica database that optimizes query performance and data compression. You can create a design in several ways:

- **Use Database Designer**, a tool that recommends a design for optimal performance.
- **Manually create a design**
- Use Database Designer to create an initial design and then manually modify it.

Database Designer can help you minimize how much time you spend on manual database tuning. You can also use Database Designer to redesign the database incrementally as requirements such as workloads change over time.

Database Designer runs as a background process. This is useful if you have a large design that you want to run overnight. An active SSH session is not required, so design and deploy operations continue to run uninterrupted if the session ends.

**Tip:** Vertica recommends that you first globally optimize your database using the Comprehensive setting in Database Designer. If the performance of the comprehensive design is not adequate, you can design custom projections using an incremental design and manually, as described in Creating Custom Designs.

About Database Designer

Vertica Database Designer uses sophisticated strategies to create a design that provides excellent performance for ad-hoc queries and specific queries while using disk space efficiently.

During the design process, Database Designer analyzes the logical schema definition, sample data, and sample queries, and creates a physical schema (projections) in the form of a SQL script that you deploy automatically or manually. This script creates a minimal set of superprojections to ensure K-safety.
In most cases, the projections that Database Designer creates provide excellent query performance within physical constraints while using disk space efficiently.

General Design Options

When you run Database Designer, several general options are available:

- Create a comprehensive or incremental design.
- Optimize for query execution, load, or a balance of both.
- Require K-safety.
- Recommend unsegmented projections when feasible.
- Analyze statistics before creating the design.

Design Input

Database Designer bases its design on the following information that you provide:

- Design queries that you typically run during normal database operations.
- Design tables that contain sample data.

Output

Database Designer yields the following output:

- A design script that creates the projections for the design in a way that meets the optimization objectives and distributes data uniformly across the cluster.

- A deployment script that creates and refreshes the projections for your design. For comprehensive designs, the deployment script contains commands that remove non-optimized projections. The deployment script includes the full design script.

- A backup script that contains SQL statements to deploy the design that existed on the system before deployment. This file is useful in case you need to revert to the pre-deployment design.
Design Restrictions

Database Designer-generated designs:

- Exclude live aggregate or Top-K projections. You must create these manually. See `CREATE PROJECTION (Live Aggregate Projections)`.

- Do not sort, segment, or partition projections on LONG VARBINARY and LONG VARCHAR columns.

Post-Design Options

While running Database Designer, you can choose to deploy your design automatically after the deployment script is created, or to deploy it manually, after you have reviewed and tested the design. Vertica recommends that you test the design on a non-production server before deploying the design to your production server.

How Database Designer Creates a Design

Design Recommendations

Database Designer-generated designs can include the following recommendations:

- Sort buddy projections in the same order, which can significantly improve load, recovery, and site node performance. All buddy projections have the same base name so that they can be identified as a group.

  Note: If you manually create projections, Database Designer recommends a buddy with the same sort order, if one does not already exist. By default, Database Designer recommends both super and non-super segmented projections with a buddy of the same sort order and segmentation.

- Accepts unlimited queries for a comprehensive design.

- Allows you to analyze column correlations. Correlation analysis typically only needs to be performed once and only if the table has more than DBDCorrelationSampleRowCount
(default: 4000) rows.

By default, Database Designer does not analyze column correlations. To set the correlation analysis mode, use `DESIGNER_SET_ANALYZE_CORRELATIONS_MODE`

- Identifies similar design queries and assigns them a signature.

  For queries with the same signature, Database Designer weights the queries, depending on how many queries have that signature. It then considers the weighted query when creating a design.

- Recommends and creates projections in a way that minimizes data skew by distributing data uniformly across the cluster.

- Produces higher quality designs by considering UPDATE, DELETE, and SELECT statements.

**Who Can Run Database Designer**

To use Administration Tools to run Database Designer and create an optimal database design, you must be a DBADMIN user.

To run Database Designer programmatically or using Management Console, you must be one of two types of users:

- DBADMIN user

- Have been assigned the DBDUSER role and be the owner of database tables for which you are creating a design

**Granting and Enabling the DBDUSER Role**

For a non-DBADMIN user to be able to run Database Designer using Management Console, follow the steps described in [Allowing the DBDUSER to Run Database Designer Using Management Console](#).

For a non-DBADMIN user to be able to run Database Designer programmatically, following the steps described in [Allowing the DBDUSER to Run Database Designer Programmatically](#).

**Important**: When you grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs. (For instructions about how to associate a resource pool with a user, see [User Profiles](#).)
Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Management Console or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into system resource pools for less-intensive tasks.

Allowing the DBDUSER to Run Database Designer Using Management Console

To allow a user with the DBDUSER role to run Database Designer using Management Console, you must create the user on the Vertica server.

As DBADMIN, take these steps on the server:

1. Add a temporary folder to all cluster nodes.

   `=> CREATE LOCATION '/tmp/dbd' ALL NODES;`

2. Create the user who needs access to Database Designer.

   `=> CREATE USER new_user;`

3. Grant the user the privilege to create schemas on the database for which they want to create a design.

   `=> GRANT CREATE ON DATABASE new_database TO new_user;`

4. Grant the DBDUSER role to the new user.

   `=> GRANT DBDUSER TO new_user;`

5. On all nodes in the cluster, grant the user access to the temporary folder.

   `=> GRANT ALL ON LOCATION '/tmp/dbd' TO new_user;`

6. Grant the new user access to the database schema and its tables.

   `=> GRANT ALL ON SCHEMA user_schema TO new_user;`
   `=> GRANT ALL ON ALL TABLES IN SCHEMA user_schema TO new_user;`

After you have completed this task, map the MC user to `new_user:`
1. Log in to Management Console as an MC Super user.

2. Click MC Settings.

3. Click User Management.

4. To create a new MC user, click Add. To use an existing MC user, select the user and click Edit.

5. Next to the DB access level window, click Add.

6. In the Add Permissions window, do the following:
   a. From the Choose a database drop-down list, select the database for which you want the user to be able to create a design.
   b. In the Database username field, enter the user name you created on the Vertica server, new_user in this example.
   c. In the Database password field, enter the database password.
   d. In the Restrict access drop-down list, select the level of MC user you want for this user.

7. Click OK to save your changes.

8. Log out of the MC Super user account.

The MC user is now mapped to the user that you created on the Vertica server. Log in as the MC user and use Database Designer to create an optimized design for your database.

For more information about MC users, see About MC Users.

Allowing the DBDUSER to Run Database Designer Programmatically

To allow a user with the DBDUSER role to run Database Designer programmatically, take these steps:

1. The DBADMIN user must grant the DBDUSER role:

   ```
   => GRANT DBDUSER TO <username>;
   ```

   This role persists until the DBADMIN user revokes it.

2. For a non-DBADMIN user to run the Database Designer programmatically or using Management Console, one of the following two steps must happen first:
If the user's default role is already DBDUSER, skip this step. Otherwise, The user must enable the DBDUSER role:

```sql
=> SET ROLE DBDUSER;
```

The DBADMIN must add DBDUSER as the default role for that user:

```sql
=> ALTER USER <username> DEFAULT ROLE DBDUSER;
```

## DBDUSER Capabilities and Limitations

The DBDUSER role has the following capabilities and limitations:

- A DBDUSER cannot create a design with a K-safety less than the system K-safety. If the designs violate the current K-safety by not having enough buddy projections for the tables, the design does not complete.

- A DBDUSER cannot explicitly change the ancient history mark (AHM), even during deployment of their design.

When you create a design, you automatically have privileges to manipulate the design. Other tasks may require that the DBDUSER have additional privileges:

<table>
<thead>
<tr>
<th>To...</th>
<th>DBDUSER must have...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add design tables</td>
<td>• USAGE privilege on the design table schema&lt;br&gt;• OWNER privilege on the design table</td>
</tr>
<tr>
<td>Add a single design query</td>
<td>• Privilege to execute the design query</td>
</tr>
<tr>
<td>Add a file of design queries</td>
<td>• Read privilege on the storage location that contains the query file&lt;br&gt;• Privilege to execute all the queries in the file</td>
</tr>
<tr>
<td>Add design queries from the result of a user query</td>
<td>• Privilege to execute the user query&lt;br&gt;• Privilege to execute each design query retrieved from the results of the user query</td>
</tr>
<tr>
<td>Create the design and deployment scripts</td>
<td>• WRITE privilege on the storage location of the design script</td>
</tr>
<tr>
<td>To...</td>
<td>DBDUSER must have...</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>• WRITE privilege on the storage location of the deployment script</td>
</tr>
</tbody>
</table>

**Workflow for Running Database Designer**

Vertica provides three ways to run Database Designer:

- Creating a Database Design in Management Console
- Using Administration Tools to Create a Design
- Running Database Designer Programmatically

The following workflow is common to all these ways to run Database Designer:
Logging Projection Data for Database Designer

When you run Database Designer, the Optimizer proposes a set of ideal projections based on the options that you specify. When you deploy the design, Database Designer creates the design based on these projections. However, space or budget constraints may prevent Database Designer from creating all the proposed projections. In addition, Database Designer may not be able to implement the projections using ideal criteria.

To get information about the projections, first enable the Database Designer logging capability. When enabled, Database Designer stores information about the proposed projections in two Data Collector tables. After Database Designer deploys the design, these logs contain information about which proposed projections were actually created. After deployment, the logs contain information about:

- Projections that the Optimizer proposed
- Projections that Database Designer actually created when the design was deployed
- Projections that Database Designer created, but not with the ideal criteria that the Optimizer identified.
- The DDL used to create all the projections
- Column optimizations

If you do not deploy the design immediately, review the log to determine if you want to make any changes. If the design has been deployed, you can still manually create some of the projections that Database Designer did not create.

To enable the Database Designer logging capability, see Enabling Logging for Database Designer

To view the logged information, see Viewing Database Designer Logs.

Enabling Logging for Database Designer

By default, Database Designer does not log information about the projections that the Optimizer proposed and the Database Designer deploys.

To enable Database Designer logging, enter the following command:

```
=> ALTER DATABASE mydb SET DBDLogInternalDesignProcess = 1;
```

To disable Database Designer logging, enter the following command:
ALTER DATABASE mydb SET DBDLogInternalDesignProcess = 0;

See Also

- Logging Projection Data for Database Designer
- Viewing Database Designer Logs

Viewing Database Designer Logs

You can find data about the projections that Database Designer considered and deployed in two Data Collector tables:

- DC_DESIGN_PROJECTION_CANDIDATES
- DC_DESIGN_QUERY_PROJECTION_CANDIDATES

DC_DESIGN_PROJECTION_CANDIDATES

The DC_DESIGN_PROJECTION_CANDIDATES table contains information about all the projections that the Optimizer proposed. This table also includes the DDL that creates them. The is_a_winner field indicates if that projection was part of the actual deployed design. To view the DC_DESIGN_PROJECTION_CANDIDATES table, enter:

```
=> SELECT * FROM DC_DESIGN_PROJECTION_CANDIDATES;
```

DC_DESIGN_QUERY_PROJECTION_CANDIDATES

The DC_DESIGN_QUERY_PROJECTION_CANDIDATES table lists plan features for all design queries.

Possible features are:

- FULLY DISTRIBUTED JOIN
- MERGE JOIN
- GROUPBY PIPE
- FULLY DISTRIBUTED GROUPBY
- RLE PREDICATE
• VALUE_INDEX_Predicate

• LATE_MATERIALIZATION

For all design queries, the DC_DATA_QUERY_PROJECTION_CANDIDATES table includes the following plan feature information:

• Optimizer path cost.

• Database Designer benefits.

• Ideal plan feature and its description, which identifies how the referenced projection should be optimized.

• If the design was deployed, the actual plan feature and its description is included in the table. This information identifies how the referenced projection was actually optimized.

Because most projections have multiple optimizations, each projection usually has multiple rows. To view the DC_DATA_QUERY_PROJECTION_CANDIDATES table, enter:

```sql
=> SELECT * FROM DC_DATA_QUERY_PROJECTION_CANDIDATES;
```

To see example data from these tables, see Database Designer Logs: Example Data.

Database Designer Logs: Example Data

In the following example, Database Designer created the logs after creating a comprehensive design for the VMart sample database. The output shows two records from the DC_DATA_QUERY_PROJECTION_CANDIDATES table.

The first record contains information about the customer_dimension_dbd_1_sort_$_customer_gender$_$_annual_income$ projection. The record includes the CREATE PROJECTION statement that Database Designer used to create the projection. The is_a_winner column is t, indicating that Database Designer created this projection when it deployed the design.

The second record contains information about the product_dimension_dbd_2_sort_$_product_version$_$_product_key$ projection. For this projection, the is_a_winner column is f. The Optimizer recommended that Database Designer create this projection as part of the design. However, Database Designer did not create the projection when it deployed the design. The log includes the DDL for the CREATE PROJECTION statement. If you want to add the projection manually, you can use that DDL. For more information, see Creating a Design Manually.
```sql
=> SELECT * FROM dc_design_projection_candidates;
- [ RECORD 1 ]---------------------------------------------------------------
time | 2014-04-11 06:30:17.918764-07
node_name | v_vmart_node0001
session_id | localhost.localdoma-931:0x1b7
user_id | 45035996273704962
user_name | dbadmin
design_id | 45035996273705182
design_table_id | 45035996273720620
projection_id | 45035996273726626
iteration_number | 1
projection_name | customer_dimension_dbd_1_sort__customer_gender__$annual_income$
projection_statement | CREATE PROJECTION v_dbd_sarahtest_sarahtest."customer_dimension_dbd_1_
 sort__$customer_gender__$annual_income$"

( customer_key ENCODING AUTO,
customer_type ENCODING AUTO,
customer_name ENCODING AUTO,
customer_gender ENCODING RLE,
title ENCODING AUTO,
household_id ENCODING AUTO,
customer_address ENCODING AUTO,
customer_city ENCODING AUTO,
customer_state ENCODING AUTO,
customer_region ENCODING AUTO,
marital_status ENCODING AUTO,
customer_age ENCODING AUTO,
number_of_children ENCODING AUTO,
annual_income ENCODING AUTO,
occupation ENCODING AUTO,
largest_bill_amount ENCODING AUTO,
store_membership_card ENCODING AUTO,
customer_since ENCODING AUTO,
deal_stage ENCODING AUTO,
deal_size ENCODING AUTO,
last_deal_update ENCODING AUTO
)
AS
SELECT customer_key,
customer_type,
customer_name,
customer_gender,
title,
household_id,
customer_address,
customer_city,
customer_state,
customer_region,
marital_status,
customer_age,
number_of_children,
annual_income,
occupation,
largest_bill_amount,
store_membership_card,
customer_since,
deal_stage,
deal_size,
last_deal_update
FROM public.customer_dimension
```
ORDER BY customer_gender, annual_income
UNSEGMENTED ALL NODES;
is_a_winner | t
-[ RECORD 2 ]-----------------------------------------------
    time | 2014-04-11 06:30:17.961324-07
node_name | v_vmart_node0001
session_id | localhost.localdomain-931:0x1b7
user_id | 45035996273704962
user_name | dbadmin
design_id | 45035996273705182
design_table_id | 45035996273720624
projection_id | 45035996273726714
iteration_number | 1
projection_name | product_dimension_dbd_2_sort__$product_version$__$product_key$
projection_statement | CREATE PROJECTION v_dbd_sarahtest_sarahtest."product_dimension_dbd_2_sort__$product_version$__$product_key$
                        (product_key ENCODING AUTO, product_version ENCODING RLE, product_description ENCODING AUTO, sku_number ENCODING AUTO, category_description ENCODING AUTO, department_description ENCODING AUTO, package_type_description ENCODING AUTO, package_size ENCODING AUTO, fat_content ENCODING AUTO, diet_type ENCODING AUTO, weight ENCODING AUTO, weight_units_of_measure ENCODING AUTO, shelf_width ENCODING AUTO, shelf_height ENCODING AUTO, shelf_depth ENCODING AUTO, product_price ENCODING AUTO, product_cost ENCODING AUTO, lowest_competitor_price ENCODING AUTO, highest_competitor_price ENCODING AUTO, average_competitor_price ENCODING AUTO, discontinued_flag ENCODING AUTO )
) AS
SELECT product_key, product_version, product_description, sku_number, category_description, department_description, package_type_description, package_size, fat_content, diet_type, weight, weight_units_of_measure, shelf_width, shelf_height, shelf_depth, product_price, product_cost, lowest_competitor_price, highest_competitor_price,
average_competitor_price,  
discontinued_flag  
FROM public.product_dimension  
ORDER BY product_version,  
product_key  
UNSEGMENTED ALL NODES;  
is_a_winner | f  
.  
.  
.  

The next example shows the contents of two records in the DC_DESIGN_QUERY_PROJECTION_CANDIDATES. Both of these rows apply to projection id 45035996273726626.

In the first record, the Optimizer recommends that Database Designer optimize the customer_gender column for the GROUPBY PIPE algorithm.

In the second record, the Optimizer recommends that Database Designer optimize the public.customer_dimension table for late materialization. Late materialization can improve the performance of joins that might spill to disk.

```sql
=> SELECT * FROM dc_design_query_projection_candidates;

<table>
<thead>
<tr>
<th>time</th>
<th>2014-04-11 06:30:17.482377-07</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>session_id</td>
<td>localhost.localdoma-931:0x1b7</td>
</tr>
<tr>
<td>user_id</td>
<td>45035996273704962</td>
</tr>
<tr>
<td>user_name</td>
<td>dbadmin</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705182</td>
</tr>
<tr>
<td>design_query_id</td>
<td>3</td>
</tr>
<tr>
<td>iteration_number</td>
<td>1</td>
</tr>
<tr>
<td>design_table_id</td>
<td>45035996273720620</td>
</tr>
<tr>
<td>projection_id</td>
<td>45035996273726626</td>
</tr>
<tr>
<td>ideal_plan_feature</td>
<td>GROUP BY PIPE</td>
</tr>
<tr>
<td>ideal_plan_feature_description</td>
<td>Group-by pipelined on column(s) customer_gender</td>
</tr>
<tr>
<td>dbd_benefits</td>
<td>5</td>
</tr>
<tr>
<td>opt_path_cost</td>
<td>211</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time</th>
<th>2014-04-11 06:30:17.48276-07</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>session_id</td>
<td>localhost.localdoma-931:0x1b7</td>
</tr>
<tr>
<td>user_id</td>
<td>45035996273704962</td>
</tr>
<tr>
<td>user_name</td>
<td>dbadmin</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705182</td>
</tr>
<tr>
<td>design_query_id</td>
<td>3</td>
</tr>
<tr>
<td>iteration_number</td>
<td>1</td>
</tr>
<tr>
<td>design_table_id</td>
<td>45035996273720620</td>
</tr>
<tr>
<td>projection_id</td>
<td>45035996273726626</td>
</tr>
<tr>
<td>ideal_plan_feature</td>
<td>LATE MATERIALIZATION</td>
</tr>
<tr>
<td>ideal_plan_feature_description</td>
<td>Late materialization on table public.customer_dimension</td>
</tr>
<tr>
<td>dbd_benefits</td>
<td>4</td>
</tr>
<tr>
<td>opt_path_cost</td>
<td>669</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
You can view the actual plan features that Database Designer implemented for the projections it created. To do so, query the V_INTERNAL.DC_DESIGN_QUERY_PROJECTIONS table:

```sql
=> select * from v_internal.dc_design_query_projections;
-[ RECORD 1 ]----------------------------------------------------------
time | 2014-04-11 06:31:41.19199-07
node_name | v_vmart_node0001
session_id | localhost.localdoma-931:0x1b7
user_id | 45035996273704962
user_name | dbadmin
design_id | 45035996273705182
projection_id | 2
actual_plan_id | 45035996273720624
actual_plan_feature | RLE PREDICATE
actual_plan_feature_description | RLE on predicate column(s) department_description
dbd_benefits | 2
opt_path_cost | 141

-[ RECORD 2 ]----------------------------------------------------------
time | 2014-04-11 06:31:41.192292-07
node_name | v_vmart_node0001
session_id | localhost.localdoma-931:0x1b7
user_id | 45035996273704962
user_name | dbadmin
design_id | 45035996273705182
design_query_id | 1
projection_id | 2
design_table_id | 45035996273720624
actual_plan_feature | GROUP BY PIPE
actual_plan_feature_description | Group-by pipelined on column(s) fat_content
dbd_benefits | 5
opt_path_cost | 155
```

### Specifying Parameters for Database Designer

Before you run Database Designer to create a design, provide information that allows Database Designer to create the optimal physical schema:

- **Design Name**
- **Design Types**
- **Optimization Objectives**
- **Design Tables with Sample Data**
- **Design Queries**
- **K-safety**
• Replicated and Segmented Projections

• Statistics Analysis

Design Name

All designs that Database Designer creates must have a name that you specify. The design name must be alphanumeric or underscore (_) characters, and can be no more than 32 characters long. (Administrative Tools and Management Console limit the design name to 16 characters.)

The design name becomes part of the files that Database Designer generates, including the deployment script, allowing the files to be easily associated with a particular Database Designer run.

Design Types

The Database Designer can create two distinct design types. The design you choose depends on what you are trying to accomplish:

• Comprehensive Design

• Incremental Design

Comprehensive Design

A comprehensive design creates an initial or replacement design for all the tables in the specified schemas. Create a comprehensive design when you are creating a new database.

To help Database Designer create an efficient design, load representative data into the tables before you begin the design process. When you load data into a table, Vertica creates an unoptimized superprojection so that Database Designer has projections to optimize. If a table has no data, Database Designer cannot optimize it.

Optionally, supply Database Designer with representative queries that you plan to use so Database Designer can optimize the design for them. If you do not supply any queries, Database Designer creates a generic optimization of the superprojections that minimizes storage, with no query-specific projections.

During a comprehensive design, Database Designer creates deployment scripts that:
Create new projections to optimize query performance, only when they do not already exist.

Create replacement buddy projections when Database Designer changes the encoding of pre-existing projections that it has decided to keep.

Incremental Design

After you create and deploy a comprehensive database design, it's likely that your database will change over time in various ways. You should periodically consider using Database Designer to create incremental designs that address these changes. Changes that warrant an incremental design can include:

- Significant data additions or updates
- New or modified queries that you run regularly
- Performance issues with one or more queries
- Schema changes

Optimization Objectives

When creating a design, Database Designer can optimize the design for one of three objectives:

- **Load**: Database Designer creates a design that is optimized for loads, minimizing database size, potentially at the expense of query performance.

- **Performance**: Database Designer creates a design that is optimized for fast query performance. Because it recommends a design for optimized query performance, this design might recommend more than the Load or Balanced objectives, potentially resulting in a larger database storage size.

- **Balanced**: Database Designer creates a design whose objectives are balanced between database size and query performance.

A fully optimized query has an optimization ratio of 0.99. Optimization ratio is the ratio of a query's benefits achieved in the design produced by the Database Designer to that achieved in the ideal plan. Check the optimization ratio with the OptRatio parameter in designer.log.
Design Tables with Sample Data

You must specify one or more design tables for Database Designer to deploy a design. If your schema is empty, it does not appear as a design table option.

When you specify design tables, consider the following:

- To create the most efficient projections for your database, load a moderate amount of representative data into tables before running Database Designer. Database Designer considers the data in this table when creating the design.

- If your design tables have a large amount of data, the Database Designer run takes a long time; if your tables have too little data, the design is not optimized. Vertica recommends that 10 GB of sample data is sufficient for creating an optimal design.

- If you submit a design table with no data, Database Designer ignores it.

- If one of your design tables has been dropped, you will not be able to build or deploy your design.

Design Queries

If you supply representative queries that you run on your database to Database Designer, it optimizes the performance of those queries.

Database Designer checks the validity of all queries when you add them to your design and again when it builds the design. If a query is invalid, Database Designer ignores it.

The query file can contain up to 100 queries. Each query can be assigned a weight that indicates its relative importance so that Database Designer can prioritize it when creating the design. Database Designer groups queries that affect the design that Database Designer creates in the same way and considers one weighted query when creating a design.

The following options apply, depending on whether you create an incremental or comprehensive design:

- Design queries are required for incremental designs.

- Design queries are optional for comprehensive designs. If you do not provide design queries, Database Designer recommends a generic design that does not consider specific queries.
Query Repository

Using Management Console, you can submit design queries from the QUERY_REQUESTS system table. This is called the query repository.

The QUERY_REQUESTS table contains queries that users have run recently. For a comprehensive design, you can submit up to 200 queries from the QUERY_REQUESTS table to Database Designer to be considered when creating the design. For an incremental design, you can submit up to 100 queries from the QUERY_REQUESTS table.

Replicated and Segmented Projections

When creating a comprehensive design, Database Designer creates projections based on data statistics and queries. It also reviews the submitted design tables to decide whether projections should be segmented (distributed across the cluster nodes) or replicated (duplicated on all cluster nodes).

For detailed information, see the following sections:

- Replicated Projections
- Segmented Projections

Replicated Projections

Replication occurs when Vertica stores identical copies of data across all nodes in a cluster.

If you are running on a single-node database, all projections are replicated because segmentation is not possible in a single-node database.

Assuming that largest-row-count equals the number of rows in the design table with the largest number of rows, Database Designer recommends that a projection be replicated if any of the following conditions is true:

- largest-row-count < 1,000,000 and number of rows in the table <= 10% of largest-row-count
- largest-row-count >= 10,000,000 and number of rows in the table <= 1% of largest-row-count
- The number of rows in the table <= 100,000

For more information about replication, see High Availability With Projections in Vertica Concepts.
Segmented Projections

*Segmentation* occurs when Vertica distributes data evenly across multiple database nodes so that all nodes participate in query execution. Projection segmentation provides high availability and recovery, and optimizes query execution.

When running Database Designer programmatically or using Management Console, you can specify to allow Database Designer to recommend unsegmented projections in the design. If you do not specify this, Database Designer recommends only segmented projections.

Database Designer recommends segmented superprojections for large tables when deploying to multiple node clusters, and recommends replicated superprojections for smaller tables.

Database Designer does not segment projections on:

- Single-node clusters
- LONG VARCHAR and LONG VARBINARY columns

For more information about segmentation, see *High Availability With Projections* in Vertica Concepts.

Statistics Analysis

By default, Database Designer analyzes statistics for the design tables when adding them to the design. This option is optional, but Vertica recommends that you analyze statistics because accurate statistics help Database Designer optimize compression and query performance.

Analyzing statistics takes time and resources. If the current statistics for the design tables are up to date, do not bother analyzing the statistics. When in doubt, analyze the statistics to make sure they are current.

For more information, see *Collecting Statistics*.

Building a Design

After you have created design tables and loaded data into them, and then specified the parameters you want Database Designer to use when creating the physical schema, direct Database Designer to create the scripts necessary to build the design.

*Note:* You cannot stop a running database if Database Designer is building a database design.
When you build a database design, Vertica generates two scripts:

- **Deployment script:** `design-name_deploy.sql`—Contains the SQL statements that create projections for the design you are deploying, deploy the design, and drop unused projections. When the deployment script runs, it creates the optimized design. For details about how to run this script and deploy the design, see Deploying a Design.

- **Design script:** `design-name_design.sql`—Contains the CREATE PROJECTION statements that Database Designer uses to create the design. Review this script to make sure you are happy with the design.

The design script is a subset of the deployment script. It serves as a backup of the DDL for the projections that the deployment script creates.

When you create a design using Management Console:

- If you submit a large number of queries to your design and build it right immediately, a timing issue could cause the queries not to load before deployment starts. If this occurs, you might see one of the following errors:
  - No queries to optimize for
  - No tables to design projections for

To accommodate this timing issue, you may need to reset the design, check the Queries tab to make sure the queries have been loaded, and then rebuild the design. Detailed instructions are in:

  - Using the Wizard to Create a Design
  - Creating a Design Manually

- The scripts are deleted when deployment completes. To save a copy of the deployment script after the design is built but before the deployment completes, go to the Output window and copy and paste the SQL statements to a file.

### Resetting a Design

You must reset a design when:
• You build a design and the output scripts described in Building a Design are not created.

• You build a design but Database Designer cannot complete the design because the queries it expects are not loaded.

Resetting a design discards all the run-specific information of the previous Database Designer build, but retains its configuration (design type, optimization objectives, K-safety, etc.) and tables and queries.

After you reset a design, review the design to see what changes you need to make. For example, you can fix errors, change parameters, or check for and add additional tables or queries. Then you can rebuild the design.

You can only reset a design in Management Console or by using the DESIGNER_RESET_DESIGN function.
Deploying a Design

After running Database Designer to generate a deployment script, Vertica recommends that you test your design on a non-production server before you deploy it to your production server.

Both the design and deployment processes run in the background. This is useful if you have a large design that you want to run overnight. Because an active SSH session is not required, the design/deploy operations continue to run uninterrupted, even if the session is terminated.

**Note:** You cannot stop a running database if Database Designer is building or deploying a database design.

Database Designer runs as a background process. Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. However, if multiple users are deploying a design on the same tables at the same time, Database Designer may not be able to complete the deployment. To avoid problems, consider the following:

- Schedule potentially conflicting Database Designer processes to run sequentially overnight so that there are no concurrency problems.

- Avoid scheduling Database Designer runs on the same set of tables at the same time.

There are two ways to deploy your design:

- **Deploying Designs Using Database Designer**

- **Deploying Designs Manually**

Deploying Designs Using Database Designer

[[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you run Database Designer and deploy optimized projections right after loading your tables with sample data because Database Designer provides projections optimized for the current state of your database.

If you choose to allow Database Designer to automatically deploy your script during a comprehensive design and are running Administrative Tools, Database Designer creates a backup script of your database's current design. This script helps you re-create the design of
projections that may have been dropped by the new design. The backup script is located in the output directory you specified during the design process.

If you choose not to have Database Designer automatically run the deployment script (for example, if you want to maintain projections from a pre-existing deployment), you can manually run the deployment script later. See Deploying Designs Manually.

To deploy a design while running Database Designer, do one of the following:

- In Management Console, select the design and click Deploy Design.
- In the Administration Tools, select Deploy design in the Design Options window.

If you are running Database Designer programmatically, use DESIGNER_RUN_POPULATE DESIGN_AND_DEPLOY and set the deploy parameter to 'true'.

Once you have deployed your design, query the DEPLOY_STATUS system table to see the steps that the deployment took:

```
vmartdb=> SELECT * FROM V_MONITOR.DEPLOY_STATUS;
```

### Deploying Designs Manually

If you choose not to have Database Designer deploy your design at design time, you can deploy the design later using the deployment script:

1. Make sure that you have a database that contains the same tables and projections as the database on which you ran Database Designer. The database should also contain sample data.

2. To deploy the projections to a test or production environment, use the following vsql command to execute the deployment script, where `design-name` is the name of the database design:

   ```
   => \i design-name_deploy.sql
   ```

### How to Create a Design

There are three ways to create a design using Database Designer:
• From Management Console, open a database and select the Design page at the bottom of the window.

For details about using Management Console to create a design, see Creating a Database Design in Management Console

• Programmatically, using the techniques described in About Running Database Designer Programmatically in Analyzing Data. To run Database Designer programmatically, you must be a DBADMIN or have been granted the DBDUSER role and enabled that role.

• From the Administration Tools menu, by selecting Configuration Menu > Run Database Designer. You must be a DBADMIN user to run Database Designer from the Administration Tools.

For details about using Administration Tools to create a design, see Using Administration Tools to Create a Design.

The following table shows what Database Designer capabilities are available in each tool:

<table>
<thead>
<tr>
<th>Database Designer Capability</th>
<th>Management Console</th>
<th>Running Database Designer Programmatically</th>
<th>Administrative Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create design</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design name length (# of characters)</td>
<td>16</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>Build design (create design and deployment scripts)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Create backup script</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Set design type (comprehensive or incremental)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Set optimization objective</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Add design tables</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Add design queries file</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Database Designer Capability</td>
<td>Management Console</td>
<td>Running Database Designer Programmatically</td>
<td>Administrative Tools</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Add single design query</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Use query repository</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Set K-safety</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Analyze statistics</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Require all unsegmented projections</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>View event history</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Set correlation analysis mode (Default = 0)</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Using Administration Tools to Create a Design**

To use the Administration Tools interface to create an optimized design for your database, you must be a DBADMIN user. Follow these steps:

1. Log in as the dbadmin user and start Administration Tools.

2. From the main menu, start the database for which you want to create a design. The database must be running before you can create a design for it.

3. On the main menu, select Configuration Menu and click OK.

4. On the Configuration Menu, select Run Database Designer and click OK.

5. On the Select a database to design window, enter the name of the database for which you are creating a design and click OK.

6. On the Enter the directory for Database Designer output window, enter the full path to the directory to contain the design script, deployment script, backup script, and log files, and click OK.

   For information about the scripts, see Building a Design.
7. On the Database Designer window, enter a name for the design and click OK.
   
   For more information about design names, see Design Name.

8. On the Design Type window, choose which type of design to create and click OK.
   
   For a description of the design types, see Design Types.

9. The Select schema(s) to add to query search path window lists all the schemas in the database that you selected. Select the schemas that contain representative data that you want Database Designer to consider when creating the design and click OK.
   
   For more information about choosing schema and tables to submit to Database Designer, see Design Tables with Sample Data.

10. On the Optimization Objectives window, select the objective you want for the database optimization:
    
    - Optimize with Queries
      
      For more information, see Design Queries.
    
    - Update statistics
      
      For more information see Statistics Analysis.
    
    - Deploy design
      
      For more information, see Deploying a Design.

    For details about these objectives, see Optimization Objectives.

11. The final window summarizes the choices you have made and offers you two choices:
    
    - Proceed with building the design, and deploying it if you specified to deploy it immediately. If you did not specify to deploy, you can review the design and deployment scripts and deploy them manually, as described in Deploying Designs Manually.
    
    - Cancel the design and go back to change some of the parameters as needed.

12. Creating a design can take a long time. To cancel a running design from the Administration Tools window, enter Ctrl+C.

    To create a design for the VMart example database, see Using Database Designer to Create a Comprehensive Design in Getting Started.
Running Database Designer Programmatically

Vertica provides a set of meta-functions that enable programmatic access to Database Designer functionality. Run Database Designer programmatically to perform the following tasks:

- Optimize performance on tables that you own.
- Create or update a design without requiring superuser or DBADMIN intervention.
- Add individual queries and tables, or add data to your design, and then rerun Database Designer to update the design based on this new information.
- Customize the design.
- Use recently executed queries to set up your database to run Database Designer automatically on a regular basis.
- Assign each design query a \textit{query weight} that indicates the importance of that query in creating the design. Assign a higher weight to queries that you run frequently so that Database Designer prioritizes those queries in creating the design.

For more details about Database Designer functions, see Database Designer Function Categories.

Database Designer Function Categories

Database Designer functions perform the following operations, generally performed in the following order:

1. Create a design.
2. Set design properties.
3. Populate a design.
4. Create design and deployment scripts.
5. Get design data.
6. Clean up.
For detailed information, see Workflow for Running Database Designer Programmatically. For information on required privileges, see Privileges for Running Database Designer Functions in the Administrator's Guide.

Create a design

DESIGNER_CREATE_DESIGN directs Database Designer to create a design.

Set design properties

The following functions let you specify properties of a particular design:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_SET_DESIGN_TYPE</td>
<td>Specifies whether the design is comprehensive or incremental.</td>
</tr>
<tr>
<td>DESIGNER_DESIGN_PROJECTION_ENCODINGS</td>
<td>Analyzes encoding in the specified projections and creates a script that implements encoding recommendations.</td>
</tr>
<tr>
<td>DESIGNER_SET_DESIGN_KSAFE</td>
<td>Sets the K-safety value for a comprehensive design.</td>
</tr>
<tr>
<td>DESIGNER_SET_OPTIMIZATION_OBJECTIVE</td>
<td>Specifies whether the design optimizes for query or load performance.</td>
</tr>
<tr>
<td>DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS</td>
<td>Enables inclusion of unsegmented projections in the design.</td>
</tr>
<tr>
<td>DESIGNER_SET_ANALYZE_CORRELATIONS_MODE</td>
<td>Determines how the design handles column correlations.</td>
</tr>
</tbody>
</table>

Populate a design

The following functions let you add tables and queries to your Database Designer design:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_ADD_DESIGN_TABLES</td>
<td>Adds the specified tables to a design.</td>
</tr>
<tr>
<td>DESIGNER_ADD_DESIGN_QUERY</td>
<td>Adds queries to the design and weights them.</td>
</tr>
<tr>
<td>DESIGNER_ADD_DESIGN_QUERIES</td>
<td></td>
</tr>
<tr>
<td>DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS</td>
<td></td>
</tr>
</tbody>
</table>
Create design and deployment scripts

The following functions populate the Database Designer workspace and create design and deployment scripts. You can also analyze statistics, deploy the design automatically, and drop the workspace after the deployment:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY</td>
<td>Populates the design and creates design and deployment scripts.</td>
</tr>
<tr>
<td>DESIGNER_WAIT_FOR_DESIGN</td>
<td>Waits for a currently running design to complete.</td>
</tr>
</tbody>
</table>

Reset a design

DESIGNER_RESET_DESIGN discards all the run-specific information of the previous Database Designer build or deployment of the specified design but retains its configuration.

Get design data

The following functions display information about projections and scripts that the Database Designer created:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS</td>
<td>Sends to standard output DDL statements that define design projections.</td>
</tr>
<tr>
<td>DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT</td>
<td>Sends to standard output a design's deployment script.</td>
</tr>
</tbody>
</table>

Clean up

The following functions cancel any running Database Designer operation or drop a Database Designer design and all its contents:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_CANCEL_POPULATE_DESIGN</td>
<td>Cancels population or deployment operation for the specified design if it is currently running.</td>
</tr>
<tr>
<td>DESIGNER_DROP_DESIGN</td>
<td>Removes the schema associated with the specified design and all its contents.</td>
</tr>
<tr>
<td>DESIGNER_DROP_ALL DESIGNS</td>
<td>Removes all Database Designer-related schemas associated with the current user.</td>
</tr>
</tbody>
</table>
Workflow for Running Database Designer Programmatically

The following example shows the steps you take to create a design by running Database Designer programmatically.

**Note:** Be sure to back up the existing design using the `EXPORT_CATALOG` function before running the Database Designer functions on an existing schema. You must explicitly back up the current design when using Database Designer to create a new comprehensive design.

Before you run this example, you should have the DBDUSER role, and you should have enabled that role using the SET ROLE DBDUSER command:

1. Create a table in the public schema:

   ```sql
   => CREATE TABLE T(
       x INT,
       y INT,
       z INT,
       u INT,
       v INT,
       w INT PRIMARY KEY
   );
   ```

2. Add data to the table:

   ```sh
   \ perl -e 'for ($i=0; $i<100000; ++$i) {printf("%d, %d, %d, %d, %d, %d\n", $i/10000, $i/100, $i/10, $i/2, $i, $i);}'
   | vsql -c "COPY T FROM STDIN DELIMITER ', ' DIRECT;"
   ```

3. Create a second table in the public schema:

   ```sql
   => CREATE TABLE T2(
       x INT,
       y INT,
       z INT,
       u INT,
       v INT,
       w INT PRIMARY KEY
   );
   ```

4. Copy the data from table T1 to table T2 and commit the changes:
5. Create a new design:

```sql
=> SELECT DESIGNER_CREATE DESIGN('my_design');
```

This command adds information to the `DESIGNS` system table in the V_MONITOR schema.

6. Add tables from the public schema to the design:

```sql
=> SELECT DESIGNER_ADD DESIGN TABLES('my_design', 'public.t);
=> SELECT DESIGNER_ADD DESIGN TABLES('my_design', 'public.t2');
```

These commands add information to the `DESIGN TABLES` system table.

7. Create a file named `queries.txt` in `/tmp/examples`, or another directory where you have READ and WRITE privileges. Add the following two queries in that file and save it. Database Designer uses these queries to create the design:

```sql
SELECT DISTINCT T2.u FROM T JOIN T2 ON T.z=T2.z-1 WHERE T2.u > 0;
SELECT DISTINCT w FROM T;
```

8. Add the queries file to the design and display the results—the numbers of accepted queries, non-design queries, and unoptimizable queries:

```sql
=> SELECT DESIGNER_ADD DESIGN QUERIES
    ('my_design', '/tmp/examples/queries.txt', 'true');
```

The results show that both queries were accepted:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of accepted queries</td>
<td>2</td>
</tr>
<tr>
<td>Number of queries referencing non-design tables</td>
<td>0</td>
</tr>
<tr>
<td>Number of unsupported queries</td>
<td>0</td>
</tr>
<tr>
<td>Number of illegal queries</td>
<td>0</td>
</tr>
</tbody>
</table>

The `DESIGNER_ADD DESIGN QUERIES` function populates the `DESIGN QUERIES` system table.

9. Set the design type to comprehensive. (This is the default.) A comprehensive design creates an initial or replacement design for all the design tables:
10. Set the optimization objective to query. This setting creates a design that focuses on faster query performance, which might recommend additional projections. These projections could result in a larger database storage footprint:

```sql
=> SELECT DESIGNER_SET_OPTIMIZATION_OBJECTIVE('my_design', 'query');
```

11. Create the design and save the design and deployment scripts in /tmp/examples, or another directory where you have READ and WRITE privileges. The following command:

- Analyzes statistics
- Doesn't deploy the design.
- Doesn't drop the design after deployment.
- Stops if it encounters an error.

```sql
=> SELECT DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY
    ('my_design',
    '/tmp/examples/my_design_projections.sql',
    '/tmp/examples/my_design_deploy.sql',
    'True',
    'False',
    'False',
    'False');
```

This command adds information to the following system tables:

- **DEPLOYMENT_PROJECTION_STATEMENTS**
- **DEPLOYMENT_PROJECTIONS**
- **OUTPUT_DEPLOYMENT_STATUS**

12. Examine the status of the Database Designer run to see what projections Database Designer recommends. In the deployment_projection_name column:

- `rep` indicates a replicated projection
- `super` indicates a superprojection
The deployment_status column is pending because the design has not yet been deployed.

For this example, Database Designer recommends four projections:

```sql
=> \x
Expanded display is on.
=> SELECT * FROM OUTPUT_DEPLOYMENT_STATUS;
- [ RECORD 1 ]-----------------------------------
deployment_id | 45035996273795970
deployment_projection_id | 1
deployment_projection_name | T_DBD_1_rep_my_design
deployment_status | pending
error_message | N/A
- [ RECORD 2 ]-----------------------------------
deployment_id | 45035996273795970
deployment_projection_id | 2
deployment_projection_name | T2_DBD_2_rep_my_design
deployment_status | pending
error_message | N/A
- [ RECORD 3 ]-----------------------------------
deployment_id | 45035996273795970
deployment_projection_id | 3
deployment_projection_name | T_super
deployment_status | pending
error_message | N/A
- [ RECORD 4 ]-----------------------------------
deployment_id | 45035996273795970
deployment_projection_id | 4
deployment_projection_name | T2_super
deployment_status | pending
error_message | N/A
```

13. View the script /tmp/examples/my_design_deploy.sql to see how these projections are created when you run the deployment script. In this example, the script also assigns the encoding schemes RLE and COMMONDELTA_COMP to columns where appropriate.

14. Deploy the design from the directory where you saved it:

```sql
=> \i /tmp/examples/my_design_deploy.sql
```

15. Now that the design is deployed, delete the design:

```sql
=> SELECT DESIGNER_DROP_DESIGN('my_design');
```

**Privileges for Running Database Designer Functions**

Non-DBADMIN users with the DBDUSER role can run Database Designer functions. Two steps are required to enable users to run these functions:
1. A DBADMIN or superuser grants the user the DBDUSER role:

=> GRANT DBDUSER TO username;

This role persists until the DBADMIN revokes it.

2. Before the DBDUSER can run Database Designer functions, one of the following must occur:

- The user enables the DBDUSER role:

  => SET ROLE DBDUSER;

- The superuser sets the user's default role to DBDUSER:

  => ALTER USER username DEFAULT ROLE DBDUSER;

General DBDUSER Limitations

As a DBDUSER, the following restrictions apply:

- You can set a design's K-safety to a value less than or equal to system K-safety. You cannot change system K-safety.

- You cannot explicitly change the ancient history mark (AHM), even during design deployment.

Design Dependencies and Privileges

Individual design tasks are likely to have dependencies that require specific privileges:

<table>
<thead>
<tr>
<th>Task</th>
<th>Required privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add tables to a design</td>
<td>• USAGE privilege on the design table schema</td>
</tr>
<tr>
<td></td>
<td>• OWNER privilege on the design table</td>
</tr>
<tr>
<td>Add a single design query to the</td>
<td>• Privilege to execute the design query</td>
</tr>
<tr>
<td>design</td>
<td></td>
</tr>
<tr>
<td>Add a query file to the design</td>
<td>• Read privilege on the storage location that contains the query file</td>
</tr>
<tr>
<td>Task</td>
<td>Required privileges</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Add queries from the result of a user query to the design | • Privilege to execute the user query  
                • Privilege to execute each design query retrieved from the results of the user query |
| Create design and deployment scripts           | • WRITE privilege on the storage location of the design script  
                • WRITE privilege on the storage location of the deployment script |

**Resource Pool for Database Designer Users**

When you grant a user the DBDUSER role, be sure to associate a resource pool with that user to manage resources during Database Designer runs. This allows multiple users to run Database Designer concurrently without interfering with each other or using up all cluster resources.

*Note:* When a user runs Database Designer, execution is mostly contained in the user's resource pool. However, Vertica might also use other system resource pools to perform less-intensive tasks.
Creating Custom Designs

Vertica strongly recommends that you use the physical schema design produced by Database Designer, which provides K-safety, excellent query performance, and efficient use of storage space. If any queries run less as efficiently than you expect, consider using the Database Designer incremental design process to optimize the database design for the query.

If the projections created by Database Designer still do not meet your needs, you can write custom projections, from scratch or based on projection designs created by Database Designer.

If you are unfamiliar with writing custom projections, start by modifying an existing design generated by Database Designer.

Custom Design Process

To create a custom design or customize an existing one:

1. Plan the new design or modifications to an existing one. See Planning Your Design.
2. Create or modify projections. See Design Fundamentals and CREATE PROJECTION for more detail.
3. Deploy projections to a test environment. See Writing and Deploying Custom Projections.
4. Test and modify projections as needed.
5. After you finalize the design, deploy projections to the production environment.
Planning Your Design

The syntax for creating a design is easy for anyone who is familiar with SQL. As with any successful project, however, a successful design requires some initial planning. Before you create your first design:

- Become familiar with standard design requirements and plan your design to include them. See Design Requirements.
- Determine how many projections you need to include in the design. See Determining the Number of Projections to Use.
- Determine the type of compression and encoding to use for columns. See Data Encoding and Compression.
- Determine whether or not you want the database to be K-safe. Vertica recommends that all production databases have a minimum K-safety of one (K=1). Valid K-safety values are 0, 1, and 2. See Using Database Designer.

Design Requirements

A physical schema design is a script that contains CREATE PROJECTION statements. These statements determine which columns are included in projections and how they are optimized.

If you use Database Designer as a starting point, it automatically creates designs that meet all fundamental design requirements. If you intend to create or modify designs manually, be aware that all designs must meet the following requirements:

- Every design must create at least one superprojection for every table in the database that is used by the client application. These projections provide complete coverage that enables users to perform ad-hoc queries as needed. They can contain joins and they are usually configured to maximize performance through sort order, compression, and encoding.
- Query-specific projections are optional. If you are satisfied with the performance provided through superprojections, you do not need to create additional projections. However, you can maximize performance by tuning for specific query work loads.
- Vertica recommends that all production databases have a minimum K-safety of one (K=1) to support high availability and recovery. (K-safety can be set to 0, 1, or 2.) See High Availability With Projections in Vertica Concepts and Using Database Designer.
Vertica recommends that if you have more than 20 nodes, but small tables, do not create replicated projections. If you create replicated projections, the catalog becomes very large and performance may degrade. Instead, consider segmenting those projections.

Determining the Number of Projections to Use

In many cases, a design that consists of a set of superprojections (and their buddies) provides satisfactory performance through compression and encoding. This is especially true if the sort orders for the projections have been used to maximize performance for one or more query predicates (WHERE clauses).

However, you might want to add additional query-specific projections to increase the performance of queries that run slowly, are used frequently, or are run as part of business-critical reporting. The number of additional projections (and their buddies) that you create should be determined by:

- Your organization's needs
- The amount of disk space you have available on each node in the cluster
- The amount of time available for loading data into the database

As the number of projections that are tuned for specific queries increases, the performance of these queries improves. However, the amount of disk space used and the amount of time required to load data increases as well. Therefore, you should create and test designs to determine the optimum number of projections for your database configuration. On average, organizations that choose to implement query-specific projections achieve optimal performance through the addition of a few query-specific projections.

Designing for K-Safety

Vertica recommends that all production databases have a minimum K-safety of one (K=1). Valid K-safety values for production databases are 1 and 2. Non-production databases do not have to be K-safe and can be set to 0.

A K-safe database must have at least three nodes, as shown in the following table:

<table>
<thead>
<tr>
<th>K-safety level</th>
<th>Number of required nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3+</td>
</tr>
<tr>
<td>2</td>
<td>5+</td>
</tr>
</tbody>
</table>
Note: Vertica only supports K-safety levels 1 and 2.

You can set K-safety to 1 or 2 only when the physical schema design meets certain redundancy requirements. See Requirements for a K-Safe Physical Schema Design.
Using Database Designer

To create designs that are K-safe, Vertica recommends that you use the Database Designer. When creating projections with Database Designer, projection definitions that meet K-safe design requirements are recommended and marked with a K-safety level. Database Designer creates a script that uses the \texttt{MARK\_DESIGN\_KSAFE} function to set the K-safety of the physical schema to 1. For example:

\begin{verbatim}
=> \i VMart_Schema_design_opt_1.sql
CREATE PROJECTION
CREATE PROJECTION
mark_design_ksafe
----------------------
Marked design 1-safe
(1 row)
\end{verbatim}

By default, Vertica creates K-safe superprojections when database K-safety is greater than 0.
Monitoring K-Safety

Monitoring tables can be accessed programmatically to enable external actions, such as alerts. You monitor the K-safety level by querying the SYSTEM table for settings in columns DESIGNED_FAULT_TOLERANCE and CURRENT_FAULT_TOLERANCE.
Loss of K-Safety

When K nodes in your cluster fail, your database continues to run, although performance is affected. Further node failures could potentially cause the database to shut down if the failed node’s data is not available from another functioning node in the cluster.
See Also

K-Safety
Requirements for a K-Safe Physical Schema Design

Database Designer automatically generates designs with a K-safety of 1 for clusters that contain at least three nodes. (If your cluster has one or two nodes, it generates designs with a K-safety of 0. You can modify a design created for a three-node (or greater) cluster, and the K-safe requirements are already set.

If you create custom projections, your physical schema design must meet the following requirements to be able to successfully recover the database in the event of a failure:

- **Segmented projections must be segmented across all nodes.** Refer to Designing for Segmentation and Designing Segmented Projections for K-Safety.

- **Replicated projections must be replicated on all nodes.** See Designing Unsegmented Projections for K-Safety.

- **Segmented projections must have K+1 buddy projections**—projections with identical columns and segmentation criteria, where corresponding segments are placed on different nodes.

You can use the `MARK_DESIGN_KSAFE` function to find out whether your schema design meets requirements for K-safety.
Requirements for a Physical Schema Design with No K-Safety

If you use Database Designer to generate an comprehensive design that you can modify and you do not want the design to be K-safe, set K-safety level to 0 (zero).

If you want to start from scratch, do the following to establish minimal projection requirements for a functioning database with no K-safety (K=0):

1. Define at least one superprojection for each table in the logical schema.
2. Replicate (define an exact copy of) each dimension table superprojection on each node.
Designing Segmented Projections for K-Safety

Projections must comply with database K-safety requirements. In general, you must create buddy projections for each segmented projection, where the number of buddy projections is K+1. Thus, if system K-safety is set to 1, each projection segment must be duplicated by one buddy; if K-safety is set to 2, each segment must be duplicated by two buddies.
Automatic Creation of Buddy Projections

You can use CREATE PROJECTION so it automatically creates the number of buddy projections required to satisfy K-safety, by including SEGMENTED BY ... ALL NODES. If CREATE PROJECTION specifies K-safety (KSAFE=n), Vertica uses that setting; if the statement omits KSAFE, Vertica uses system K-safety.

In the following example, CREATE PROJECTION creates segmented projection ttt_p1 for table ttt. Because system K-safety is set to 1, Vertica requires a buddy projection for each segmented projection. The CREATE PROJECTION statement omits KSAFE, so Vertica uses system K-safety and creates two buddy projections: ttt_p1_b0 and ttt_p1_b1:

```sql
=> SELECT mark_design_ksafe(1);
  mark_design_ksafe
----------------------
Marked design 1-safe
(1 row)
=> CREATE TABLE ttt (a int, b int);
WARNING 6978: Table "ttt" will include privileges from schema "public"
CREATE TABLE
=> CREATE PROJECTION ttt_p1 as SELECT * FROM ttt SEGMENTED BY HASH(a) ALL NODES;
CREATE PROJECTION
=> SELECT projection_name from projections WHERE anchor_table_name='ttt';
  projection_name
----------
ttt_p1_b0
ttt_p1_b1
(2 rows)
```

Vertica automatically names buddy projections by appending the suffix _bn to the projection base name—for example ttt_p1_b0.
Manual Creation of Buddy Projections

If you create a projection on a single node, and system K-safety is greater than 0, you must manually create the number of buddies required for K-safety. For example, you can create projection `xxx_p1` for table `xxx` on a single node, as follows:

```sql
=> CREATE TABLE xxx (a int, b int);
WARNING 6978: Table "xxx" will include privileges from schema "public"
CREATE TABLE
=> CREATE PROJECTION xxx_p1 AS SELECT * FROM xxx SEGMENTED BY HASH(a) NODES v_vmart_node0001;
CREATE PROJECTION
```

Because K-safety is set to 1, a single instance of this projection is not K-safe. Attempts to insert data into its anchor table `xxx` return with an error like this:

```sql
=> INSERT INTO xxx VALUES (1, 2);
ERROR 3586: Insufficient projections to answer query
DETAIL: No projections that satisfy K-safety found for table xxx
HINT: Define buddy projections for table xxx
```

In order to comply with K-safety, you must create a buddy projection for projection `xxx_p1`. For example:

```sql
=> CREATE PROJECTION xxx_p1_buddy AS SELECT * FROM xxx SEGMENTED BY HASH(a) NODES v_vmart_node0002;
CREATE PROJECTION
```

Table `xxx` now complies with K-safety and accepts DML statements such as `INSERT`:

```sql
VMart=> INSERT INTO xxx VALUES (1, 2);
OUTPUT
-------
1
(1 row)
```
See Also

For general information about segmented projections and buddies, see Projection Segmentation in Vertica Concepts. For information about designing for K-safety, see Using Database Designer and Designing for Segmentation.
Designing Unsegmented Projections for K-Safety

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are replicated without segmentation on all cluster nodes. You create these projections with a `CREATE PROJECTION` statement that includes the keywords `UNSEGMENTED ALL NODES`. These keywords specify to create identical instances of the projection on all cluster nodes.

The following example shows how to create an unsegmented projection for the table `store.store_dimension`:

```sql
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state) AS
   SELECT store_key, store_name, store_city, store_state
   FROM store.store_dimension
   UNSEGMENTED ALL NODES;

CREATE PROJECTION
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, `store.store_dimension_proj`. The keyword `ALL NODES` specifies to replicate the projection on all nodes:

```sql
=> \dj store.store_dimension_proj
```

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Owner</th>
<th>Node</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>store</code></td>
<td><code>store_dimension_proj</code></td>
<td><code>dbadmin</code></td>
<td>v_vmart_node0001</td>
<td></td>
</tr>
<tr>
<td><code>store</code></td>
<td><code>store_dimension_proj</code></td>
<td><code>dbadmin</code></td>
<td>v_vmart_node0002</td>
<td></td>
</tr>
<tr>
<td><code>store</code></td>
<td><code>store_dimension_proj</code></td>
<td><code>dbadmin</code></td>
<td>v_vmart_node0003</td>
<td></td>
</tr>
</tbody>
</table>

(3 rows)

For more information about projection name conventions, see Projection Naming.

Designing for Segmentation

You segment projections using hash segmentation. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.
Note: For detailed information about using hash segmentation in a projection, see CREATE PROJECTION in the SQL Reference Manual.

When segmenting projections, determine which columns to use to segment the projection. Choose one or more columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns are an excellent choice for hash segmentation. The columns must be unique across all the tables being used in a query.
Design Fundamentals

Although you can write custom projections from scratch, Vertica recommends that you use Database Designer to create a design to use as a starting point. This ensures that you have projections that meet basic requirements.

Writing and Deploying Custom Projections

Before you write custom projections, be sure to review the topics in Planning Your Design carefully. Failure to follow these considerations can result in non-functional projections.

To manually modify or create a projection:

1. Write a script to create the projection, using the `CREATE PROJECTION` statement.
2. Use the `\i meta-command` in `vsql` to run the script.
   
   **Note:** You must have a database loaded with a logical schema.

3. For a K-safe database, use the function `SELECT get_projections('table_name')` to verify that the projections were properly created. Good projections are noted as being "safe." This means that the projection has enough buddies to be K-safe.

4. If you added the new projection to a database that already has projections that contain data, you need to update the newly created projection to work with the existing projections. By default, the new projection is out-of-date (not available for query processing) until you refresh it.

5. Use the `MAKE_AHM_NOW` function to set the Ancient History Mark (AHM) to the greatest allowable epoch (now).

6. Use `DROP PROJECTION` to drop any previous projections that are no longer needed.

   These projections can waste disk space and reduce load speed if they remain in the database.

7. Run the `ANALYZE_STATISTICS` function on all projections in the database. This function collects and aggregates data samples and storage information from all nodes on which a projection is stored, and then writes statistics into the catalog. For example:

   ```sql
   =>SELECT ANALYZE_STATISTICS ('');
   ```
Designing Superprojections

Superprojections have the following requirements:

- They must contain every column within the table.
- For a K-safe design, superprojections must either be replicated on all nodes within the database cluster (for dimension tables) or paired with buddies and segmented across all nodes (for very large tables and medium large tables). See Physical Schema and High Availability With Projections in Vertica Concepts for an overview of projections and how they are stored. See Using Database Designer for design specifics.

To provide maximum usability, superprojections need to minimize storage requirements while maximizing query performance. To achieve this, the sort order for columns in superprojections is based on storage requirements and commonly used queries.

Sort Order Benefits

Column sort order is an important factor in minimizing storage requirements, and maximizing query performance.
Minimize Storage Requirements

Minimizing storage saves on physical resources and increases performance by reducing disk I/O. You can minimize projection storage by prioritizing low-cardinality columns in its sort order. This reduces the number of rows Vertica stores and accesses to retrieve query results.

After identifying projection sort columns, analyze their data and choose the most effective encoding method. The Vertica optimizer gives preference to columns with run-length encoding (RLE), so be sure to use it whenever appropriate. Run-length encoding replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, it is especially appropriate to use it for low-cardinality columns whose run length is large.
Maximize Query Performance

You can facilitate query performance through column sort order as follows:

- Where possible, sort order should prioritize columns with the lowest cardinality.
- Do not sort projections on columns of type LONG VARBINARY and LONG VARCHAR.
See Also

Combine RLE and Sort Order

Choosing Sort Order: Best Practices

When choosing sort orders for your projections, Vertica has several recommendations that can help you achieve maximum query performance, as illustrated in the following examples.
Combine RLE and Sort Order

When dealing with predicates on low-cardinality columns, use a combination of RLE and sorting to minimize storage requirements and maximize query performance.

Suppose you have a `students` table contain the following values and encoding types:

<table>
<thead>
<tr>
<th>Column</th>
<th># of Distinct Values</th>
<th>Encoded With</th>
</tr>
</thead>
<tbody>
<tr>
<td>gender</td>
<td>2 (M or F)</td>
<td>RLE</td>
</tr>
<tr>
<td>pass_fail</td>
<td>2 (P or F)</td>
<td>RLE</td>
</tr>
<tr>
<td>class</td>
<td>4 (freshman, sophomore, junior, or senior)</td>
<td>RLE</td>
</tr>
<tr>
<td>name</td>
<td>10000 (too many to list)</td>
<td>Auto</td>
</tr>
</tbody>
</table>

You might have queries similar to this one:

```
SELECT name FROM students WHERE gender = 'M' AND pass_fail = 'P' AND class = 'senior';
```

The fastest way to access the data is to work through the low-cardinality columns with the smallest number of distinct values before the high-cardinality columns. The following sort order minimizes storage and maximizes query performance for queries that have equality restrictions on gender, class, pass_fail, and name. Specify the ORDER BY clause of the projection as follows:

```
ORDER BY students.gender, students.pass_fail, students.class, students.name
```

In this example, the `gender` column is represented by two RLE entries, the `pass_fail` column is represented by four entries, and the `class` column is represented by 16 entries, regardless of the cardinality of the `students` table. Vertica efficiently finds the set of rows that satisfy all the predicates, resulting in a huge reduction of search effort for RLE encoded columns that occur early in the sort order. Consequently, if you use low-cardinality columns in local predicates, as in the previous example, put those columns early in the projection sort order, in increasing order of distinct cardinality (that is, in increasing order of the number of distinct values in each column).
If you sort this table with `student.class` first, you improve the performance of queries that restrict only on the `student.class` column, and you improve the compression of the `student.class` column (which contains the largest number of distinct values), but the other columns do not compress as well. Determining which projection is better depends on the specific queries in your workload, and their relative importance.

Storage savings with compression decrease as the cardinality of the column increases; however, storage savings with compression increase as the number of bytes required to store values in that column increases.
Maximize the Advantages of RLE

To maximize the advantages of RLE encoding, use it only when the average run length of a column is greater than 10 when sorted. For example, suppose you have a table with the following columns, sorted in order of cardinality from low to high:

| address.country, address.region, address.state, address.city, address.zipcode |

The zipcode column might not have 10 sorted entries in a row with the same zip code, so there is probably no advantage to run-length encoding that column, and it could make compression worse. But there are likely to be more than 10 countries in a sorted run length, so applying RLE to the country column can improve performance.
Put Lower Cardinality Column First for Functional Dependencies

In general, put columns that you use for local predicates (as in the previous example) earlier in the join order to make predicate evaluation more efficient. In addition, if a lower cardinality column is uniquely determined by a higher cardinality column (like city_id uniquely determining a state_id), it is always better to put the lower cardinality, functionally determined column earlier in the sort order than the higher cardinality column.

For example, in the following sort order, the Area_Code column is sorted before the Number column in the customer_info table:

```
ORDER BY customer_info.Area_Code, customer_info.Number, customer_info.Address
```

In the query, put the Area_Code column first, so that only the values in the Number column that start with 978 are scanned.

```sql
=> SELECT Address FROM customer_info WHERE Area_Code='978' AND Number='9780123457';
```
Sort for Merge Joins

When processing a join, the Vertica optimizer chooses from two algorithms:

- **Merge join**—If both inputs are pre-sorted on the join column, the optimizer chooses a merge join, which is faster and uses less memory.

- **Hash join**—Using the hash join algorithm, Vertica uses the smaller (inner) joined table to build an in-memory hash table on the join column. A hash join has no sort requirement, but it consumes more memory because Vertica builds a hash table with the values in the inner table. The optimizer chooses a hash join when projections are not sorted on the join columns.

If both inputs are pre-sorted, merge joins do not have to do any pre-processing, making the join perform faster. Vertica uses the term sort-merge join to refer to the case when at least one of the inputs must be sorted prior to the merge join. Vertica sorts the inner input side but only if the outer input side is already sorted on the join columns.

To give the Vertica query optimizer the option to use an efficient merge join for a particular join, create projections on both sides of the join that put the join column first in their respective projections. This is primarily important to do if both tables are so large that neither table fits into memory. If all tables that a table will be joined to can be expected to fit into memory simultaneously, the benefits of merge join over hash join are sufficiently small that it probably isn’t worth creating a projection for any one join column.
Sort on Columns in Important Queries

If you have an important query, one that you run on a regular basis, you can save time by putting the columns specified in the WHERE clause or the GROUP BY clause of that query early in the sort order.

If that query uses a high-cardinality column such as Social Security number, you may sacrifice storage by placing this column early in the sort order of a projection, but your most important query will be optimized.
Sort Columns of Equal Cardinality By Size

If you have two columns of equal cardinality, put the column that is larger first in the sort order. For example, a CHAR(20) column takes up 20 bytes, but an INTEGER column takes up 8 bytes. By putting the CHAR(20) column ahead of the INTEGER column, your projection compresses better.
Sort Foreign Key Columns First, From Low to High Distinct Cardinality

Suppose you have a fact table where the first four columns in the sort order make up a foreign key to another table. For best compression, choose a sort order for the fact table such that the foreign keys appear first, and in increasing order of distinct cardinality. Other factors also apply to the design of projections for fact tables, such as partitioning by a time dimension, if any.

In the following example, the table inventory stores inventory data, and product_key and warehouse_key are foreign keys to the product_dimension and warehouse_dimension tables:

```sql
=> CREATE TABLE inventory (  
date_key INTEGER NOT NULL,  
product_key INTEGER NOT NULL,  
warehouse_key INTEGER NOT NULL,  
... );
=> ALTER TABLE inventory  
   ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key)  
   REFERENCES warehouse_dimension(warehouse_key);  
ALTER TABLE inventory  
   ADD CONSTRAINT fk_inventory_product FOREIGN KEY(product_key)  
   REFERENCES product_dimension(product_key);
```

The inventory table should be sorted by warehouse_key and then product, since the cardinality of the warehouse_key column is probably lower than the cardinality of the product_key.

Prioritizing Column Access Speed

If you measure and set the performance of storage locations within your cluster, Vertica uses this information to determine where to store columns based on their rank. For more information, see Setting Storage Performance.
How Columns are Ranked

Vertica stores columns included in the projection sort order on the fastest available storage locations. Columns not included in the projection sort order are stored on slower disks. Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given $1000 + \#$ of sort columns.
- The remaining columns are given numbers from 1000–1, starting with 1000 and decrementing by one per column.

Vertica then stores columns on disk from the highest ranking to the lowest ranking. It places highest-ranking columns on the fastest disks and the lowest-ranking columns on the slowest disks.
Overriding Default Column Ranking

You can modify which columns are stored on fast disks by manually overriding the default ranks for these columns. To accomplish this, set the `ACCESSRANK` keyword in the column list. Make sure to use an integer that is not already being used for another column. For example, if you want to give a column the fastest access rank, use a number that is significantly higher than $1000 +$ the number of sort columns. This allows you to enter more columns over time without bumping into the access rank you set.

The following example sets column `store_key`'s access rank to 1500:

```
CREATE PROJECTION retail_sales_fact_p (  
    store_key ENCODING RLE `ACCESSRANK 1500`,  
    pos_transaction_number ENCODING RLE,  
    sales_dollar_amount,  
    cost_dollar_amount )  
AS SELECT  
    store_key,  
    pos_transaction_number,  
    sales_dollar_amount,  
    cost_dollar_amount  
FROM store.store_sales_fact  
ORDER BY store_key  
SEGMENTED BY HASH(pos_transaction_number) ALL NODES;
```
Managing Users and Privileges

Database users should have access to only the database resources they need to perform their tasks. For example, most users should be able to read data but not modify or insert new data, while other users might need more permissive access, such as the right to create and modify schemas, tables, and views, as well as rebalance nodes on a cluster and start or stop a database. It is also possible to allow certain users to grant other users access to the appropriate database resources.

Client authentication controls what database objects users can access and change in the database. To prevent unauthorized access, a superuser limits access to what is needed, granting privileges directly to users or to roles through a series of GRANT statements. Roles can then be granted to users, as well as to other roles.

This section introduces the privilege role model in Vertica and describes how to create and manage users.

See Also

- About Database Privileges
- About Database Roles
- GRANT Statements
- REVOKE Statements
About Database Users

Every Vertica database has one or more users. When users connect to a database, they must log on with valid credentials (username and password) that a superuser defined in the database.

Database users own the objects they create in a database, such as tables, procedures, and storage locations.

Note: By default, users have the right to create temporary tables in a database.

See Also

- Creating a Database User
- CREATE USER
- About MC Users
Types of Database Users

In a Vertica database, there are three types of users:

- Database administrator (DBADMIN)
- Object owner
- Everyone else (PUBLIC)

Note: External to a Vertica database, an MC administrator can create users through the Management Console and grant them database access. See About MC Users for details.

Database Administration User

When you install a new Vertica Analytic Database a database administration user with access to the following roles gets created:

- DBADMIN Role
- DBDUSER Role
- PSEUDOSUPERUSER Role

Access to these roles allows this user to perform all database operations. Assign a name to this user during installation using the --dba-user option (use -u for upgrades). For example:

```
--dba-user mydba
```

This example creates a database administration user called mydba. The username you use here must already exist on your operating system. See Installing Vertica with the Installation Script

If you do not use --dba-user during installation the database administrator user gets named DBADMIN by default.

Note: Do not confuse the DBADMIN user with the DBADMIN Role. The DBADMIN role is a set of privileges you assign to a specific user based on the user's position in your organization.
The Vertica Analytic Database Database Administration user is also called a superuser throughout the Vertica Analytic Database documentation. Do not confuse this superuser with the Linux superuser that manages the Linux operating system.

Create a Database Administration User in the Vertica Analytic Database

As the Database Administration user you can create other users with the same privileges:

1. Create a user:

```sql
=> CREATE USER DataBaseAdmin2;
CREATE USER
```

2. Grant the appropriate roles to the new user DataBaseAdmin2:

```sql
=> GRANT dbduser, dbadmin, pseudosuperuser to DataBaseAdmin2;
GRANT ROLE
```

The user DataBaseAdmin2 now has the same privileges granted to the original Database Administration user.

3. As the DataBaseAdmin2 user, enable the roles using SET ROLE:

```sql
=> \c VMart DataBaseAdmin2;
You are now connected to database "VMart" as user "DataBaseAdmin2".
=> SET ROLE dbadmin, dbduser, pseudosuperuser;
SET ROLE
```

4. Confirm the roles are enabled:

```sql
=> SHOW ENABLED ROLES;
name | setting
-----------------------------
enabled roles | dbduser, dbadmin, pseudosuperuser
```

See Also

- **DBADMIN Role**
- **PSEUDOSUPERUSER Role**
- **PUBLIC Role**
Object Owner

An object owner is the user who creates a particular database object and can perform any operation on that object. By default, only an owner (or a superuser) can act on a database object. In order to allow other users to use an object, the owner or superuser must grant privileges to those users using one of the GRANT Statements.

Note: Object owners are PUBLIC users for objects that other users own.

See About Database Privileges for more information.

PUBLIC User

All non-DBA (superuser) or object owners are PUBLIC users.

Note: Object owners are PUBLIC users for objects that other users own.

Newly-created users do not have access to schema PUBLIC by default. Make sure to GRANT USAGE ON SCHEMA PUBLIC to all users you create.

See Also

- PUBLIC Role

Creating a Database User

To create a database user:

1. From vsqi, connect to the database as a superuser.
2. Issue the CREATE USER statement with optional parameters.
3. Run a series of GRANT Statements to grant the new user privileges.

To create a user on MC, see Creating an MC User in Management Console

New User Privileges

By default, new database users have the right to create temporary tables in the database.
Newly-created users do not have access to schema PUBLIC by default. Make sure to `GRANT USAGE ON SCHEMA PUBLIC` to all users you create.

### Modifying Users

You can change information about a user, such as his or her password, by using the `ALTER USER` statement. If you want to configure a user to not have any password authentication, you can set the empty password '' in `CREATE USER` or `ALTER USER` statements, or omit the `IDENTIFIED BY` parameter in `CREATE USER`.

### Example

The following series of commands add user Fred to a database with password `password`. The second command grants USAGE privileges to Fred on the public schema:

```sql
=> CREATE USER Fred IDENTIFIED BY 'password';
=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

User names created with double-quotes are case sensitive. For example:

```sql
=> CREATE USER "FrEd1";
```

In the above example, the logon name must be an exact match. If the user name was created without double-quotes (for example, FRED1), then the user can log on as FRED1, FrEd1, fred1, and so on.

### See Also

- [Granting and Revoking Privileges](#)
- [Granting Access to Database Roles](#)

### Locking/unlocking a user's Database Access

A superuser can manually lock an existing database user's account with the `ALTER USER` statement. For example, the following command prevents user Fred from logging in to the database:

```sql
=> ALTER USER Fred ACCOUNT LOCK;
```
To grant Fred database access, use UNLOCK syntax with the ALTER USER command:

```
=> ALTER USER Fred ACCOUNT UNLOCK;
=> \c - Fred
You are now connected as user "Fred".
```

### Using CREATE USER to lock an account

Although not as common, you can create a new user with a locked account; for example, you might want to set up an account for a user who doesn't need immediate database access, as in the case of an employee who will join the company at a future date.

```
=> CREATE USER Bob ACCOUNT UNLOCK;
CREATE USER
```

CREATE USER also supports UNLOCK syntax; however, UNLOCK is the default, so you don't need to specify the keyword when you create a new user to whom you want to grant immediate database access.

### Locking an account automatically

Instead of manually locking an account, a superuser can automate account locking by setting a maximum number of failed login attempts through the CREATE PROFILE statement. See Profiles.

### Changing a User's Password

A superuser can change another user's database account, including reset a password, with the ALTER USER statement.

Making changes to a database user account with does not affect current sessions.

```
=> ALTER USER Fred IDENTIFIED BY 'newpassword';
```

In the above command, Fred's password is now newpassword.
Note: Non-DBA users can change their own passwords using the IDENTIFIED BY 'new-password' option along with the REPLACE 'old-password' clause. See ALTER USER for details.

Changing a User's MC Password

On MC, users with ADMIN or IT privileges can reset a user's non-LDAP password from the MC interface.

Non-LDAP passwords on MC are for MC access only and are not related to a user's logon credentials on the Vertica database.

1. Sign in to Management Console and navigate to MC Settings > User management.
2. Click to select the user to modify and click Edit.
3. Click Edit password and enter the new password twice.
4. Click OK and then click Save.
About Database Privileges

When a database object is created, such as a schema, table, or view, that object is assigned an owner—the person who executed the CREATE statement. By default, database administrators (superusers) or object owners are the only users who can do anything with the object.

In order to allow other users to use an object, or remove a user’s right to use an object, the authorized user must grant another user privileges on the object.

Privileges are granted (or revoked) through a collection of GRANT/REVOKE statements that assign the privilege—a type of permission that lets users perform an action on a database object, such as:

- Create a schema
- Create a table (in a schema)
- Create a view
- View (select) data
- Insert, update, or delete table data
- Drop tables or schemas
- Run procedures

Before Vertica executes a statement, it determines if the requesting user has the necessary privileges to perform the operation.

For more information about the privileges associated with these resources, see Privileges That Can Be Granted on Objects.

**Note:** Vertica logs information about each grant (grantor, grantee, privilege, and so on) in the V_CATALOG.GRANTS system table.

See Also

- **GRANT Statements**
- **REVOKE Statements**
Inherited Privileges

Inherited privileges allow you to grant privileges at the schema level. This enables privileges to be granted automatically to new tables or views in the schema. Existing tables and views are unchanged when you alter the schema to include or exclude inherited privileges. Using inherited privileges eliminates the need to apply the same privileges to each individual table or view in the schema.

To assign inherited privileges, you must be an owner of the schema or a superuser. Assign inherited privileges using the following SQL statements

- GRANT Statements
- CREATE SCHEMA
- ALTER SCHEMA
- CREATE TABLE
- ALTER TABLE
- CREATE VIEW
- ALTER VIEW

Granting Inherited Privileges from One User to Another

The following steps describe a process for user1 to enable inherited privileges to user2.

1. The database user, user1, creates a schema (schema1), and a table (table1) in schema1:

   ```
   user1=> CREATE SCHEMA schema1;
   user1=> CREATE TABLE schema1.table1 (id int);
   ```

2. User user1 grants USAGE and CREATE privileges on schema1 to user2:

   ```
   user1=> GRANT USAGE ON SCHEMA schema1 to user2;
   user1=> GRANT CREATE ON SCHEMA schema1 to user2;
   ```

3. The user2 user queries schema1.table1, but the query fails:
4. The user user1 grants SELECT ON SCHEMA privilege on schema1 to user2:

user1=> GRANT SELECT ON SCHEMA schema1 to user2;

5. Next, user1 uses ALTER TABLE to include SCHEMA privileges to table1:

user1=> ALTER TABLE schema1.table1 INCLUDE SCHEMA PRIVILEGES;

6. The user2 query now succeeds:

user2=> SELECT * FROM schema1.table1;
  id
  ...
(0 rows)

7. User 1 now uses ALTER SCHEMA to include privileges so that all tables created in schema1 inherit schema privileges:

user1=> ALTER SCHEMA schema1 DEFAULT INCLUDE PRIVILEGES;
user1=> CREATE TABLE schema1.table2 (id int);

8. With Inherited Privileges enabled, user2 can query table2 without user1 having to specifically grant privileges on table2:

user2=> SELECT * FROM schema1.table2;
  id
  ...
(0 rows)

Enable or Disable Inherited Privileges at the Database Level

Use the disableinheritedprivileges configuration parameter to enable (0) Inherited Privileges:

=> ALTER DATABASE [database name] SET disableinheritedprivileges = 0;

Use the following configuration parameter to disable (1) Inherited Privileges:

=> ALTER DATABASE [database name] SET disableinheritedprivileges = 1;
Grant Inherited Privileges

Grant inherited privileges at the schema level. When inherited privileges is enabled, all privileges granted to the schema are automatically granted to all newly created tables or views in the schema. Existing tables or views remain unchanged when you alter the schema to include or exclude inherited privileges.

By default, inherited privileges are enabled at the database level and disabled at the schema level (unless you indicate otherwise while running CREATE SCHEMA). See Enable or Disable Inherited Privileges at the Database Level for more information. To apply inherited privileges, you must meet one of the following conditions:

- Be the owner of the object
- Be a superuser

Inherit Privileges on a Schema

Use the CREATE SCHEMA or ALTER SCHEMA SQL statements to apply inherited privileges to a schema. The tables and views in that schema then inherit any privileges granted to the schema by default.

This example shows how to create a new schema with inherited privileges. The DEFAULT parameter sets the default behavior so that all new tables and views created in this schema automatically inherit the schema's privileges:

```
=> CREATE SCHEMA s1 DEFAULT INCLUDE PRIVILEGES;
```

This example shows how to modify an existing schema to enable inherited privileges:

```
=> ALTER SCHEMA s1 DEFAULT INCLUDE PRIVILEGES;
```

Note: After enabling inherited privileges on a schema, you still must grant privileges on the schema to a user or role. From the GRANT (Schema) statement, use the following parameters with inherited privileges enabled:

- SELECT
- INSERT
- UPDATE
The following message appears when you specify INCLUDE PRIVILEGES while Inherited privileges is disabled at the database level:

```
Inherited privileges are globally disabled; schema parameter is set but has no effect.
```

See [Enable or Disable Inherited Privileges at the Database Level](#) to enable Inherited Privileges at the database level.

**Inherit Privileges on a Table or Flex Table**

You can specify an individual table or flex table to inherit privileges from the schema. Use `CREATE TABLE` or `ALTER TABLE` SQL statements to enable inherited privileges for a table. The table-level flag takes priority over the schema flag, while the database knob takes priority over both.

This example shows creating a new table with inherited privileges:

```
=> CREATE TABLE s1.t1 ( x int) INCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing table to enable inherited privileges:

```
=> ALTER TABLE s1.t1 INCLUDE SCHEMA PRIVILEGES;
```

If you run `CREATE TABLE`, `CREATE TABLE LIKE`, or `CREATE TABLE AS SELECT` in a schema with inherited privileges set, the following informational warning appears:

```
=> CREATE TABLE s1.t1 ( x int);
WARNING: Table <table_name> will include privileges from schema <schema_name>
```

Note that this message does not appear when you add the INCLUDE SCHEMA PRIVILEGES statement.

**Exclude Privileges on a Table**

You can exclude a table in a schema with inherited privileges so that table does not inherit the schema's privileges. Use `CREATE TABLE` or `ALTER TABLE` SQL statements to exclude inherited
privileges for a table.

This example shows creating a new table and excluding schema privileges:

```sql
=> CREATE TABLE s1.t1 (x int) EXCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing table to exclude inherited privileges:

```sql
=> ALTER TABLE s1.t1 EXCLUDE SCHEMA PRIVILEGES;
```

Include Privileges on a View

You can specify a View to inherit privileges from the schema. Use `CREATE VIEW` or `ALTER VIEW` SQL statements to enable inherited privileges for a view.

This example shows creating a view with inherited privileges enabled:

```sql
=> CREATE VIEW view1 INCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing view to enable inherited privileges:

```sql
=> ALTER VIEW view1 INCLUDE SCHEMA PRIVILEGES;
```

Exclude Privileges on a View

You can exclude a view in a schema with inherited privileges so that view does not inherit the schema's privileges. Use `CREATE VIEW` or `ALTER VIEW` SQL statements to exclude inherited privileges for a view.

This example shows creating a new view and excluding schema privileges:

```sql
=> CREATE VIEW view1 EXCLUDE SCHEMA PRIVILEGES;
```

This example shows how to modify an existing view to exclude inherited privileges:

```sql
=> ALTER VIEW view1 EXCLUDE SCHEMA PRIVILEGES;
```

Default Privileges for All Users

To set the minimum level of privilege for all users, Vertica has the special `PUBLIC Role`, which it grants to each user automatically. This role is automatically enabled, but the database
administrator or a superuser can also grant higher privileges to users separately using GRANT statements.

The following topics discuss those higher privileges.

**Default Privileges for MC Users**

Privileges on Management Console (MC) are managed through roles, which determine a user's access to MC and to MC-managed Vertica databases through the MC interface. MC privileges do not alter or override Vertica privileges or roles. See About MC Privileges and Roles for details.

**Privileges Required for Common Database Operations**

This topic lists the required privileges for database objects in Vertica.

Unless otherwise noted, superusers can perform all of the operations shown in the following tables without any additional privilege requirements. Object owners have the necessary rights to perform operations on their own objects, by default.

**Schemas**

The PUBLIC schema is present in any newly-created Vertica database, and newly-created users have only USAGE privilege on PUBLIC. A database superuser must explicitly grant new users CREATE privileges, as well as grant them individual object privileges so the new users can create or look up objects in the PUBLIC schema.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE SCHEMA</td>
<td>CREATE privilege on database</td>
</tr>
<tr>
<td>DROP SCHEMA</td>
<td>Schema owner</td>
</tr>
<tr>
<td>ALTER SCHEMA RENAME</td>
<td>CREATE privilege on database</td>
</tr>
</tbody>
</table>

**Tables**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE</td>
<td>CREATE privilege on schema</td>
</tr>
<tr>
<td>Operation</td>
<td>Required Privileges</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Referencing sequences in the CREATE TABLE statement requires the following privileges:</td>
</tr>
<tr>
<td></td>
<td>• SELECT privilege on sequence object</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on sequence schema</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>USAGE privilege on the schema that contains the table or schema owner</td>
</tr>
<tr>
<td>TRUNCATE TABLE</td>
<td>USAGE privilege on the schema that contains the table or schema owner</td>
</tr>
<tr>
<td>ALTER TABLE ADD/DROP/RENAME/ALTER-TYPE COLUMN</td>
<td>USAGE privilege on the schema that contains the table</td>
</tr>
<tr>
<td>ALTER TABLE ADD/DROP CONSTRAINT</td>
<td>USAGE privilege on the schema that contains the table</td>
</tr>
<tr>
<td>ALTER TABLE PARTITION (REORGANIZE)</td>
<td>USAGE privilege on the schema that contains the table</td>
</tr>
<tr>
<td>ALTER TABLE RENAME</td>
<td>USAGE and CREATE privilege on the schema that contains the table</td>
</tr>
<tr>
<td>ALTER TABLE SET SCHEMA</td>
<td>• CREATE privilege on new schema</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on the old schema</td>
</tr>
<tr>
<td>SELECT</td>
<td>• SELECT privilege on table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on schema that contains the table</td>
</tr>
<tr>
<td>INSERT</td>
<td>• INSERT privilege on table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on schema that contains the table</td>
</tr>
<tr>
<td>DELETE</td>
<td>• DELETE privilege on table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on schema that contains the table</td>
</tr>
<tr>
<td></td>
<td>• SELECT privilege on the referenced table when executing a DELETE statement that references table column values in a WHERE or SET clause</td>
</tr>
<tr>
<td>UPDATE</td>
<td>• UPDATE privilege on table</td>
</tr>
<tr>
<td>Operation</td>
<td>Required Privileges</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>• REFERENCES privilege on table to create foreign key constraints that reference this table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privileges on schema that contains the constrained table and the source of the foreign key</td>
</tr>
<tr>
<td>ANALYZE_STATISTICS</td>
<td>• INSERT/UPDATE/DELETE privilege on table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on schema that contains the table</td>
</tr>
<tr>
<td>DROP_STATISTICS</td>
<td>• INSERT/UPDATE/DELETE privilege on table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on schema that contains the table</td>
</tr>
<tr>
<td>DROP_PARTITIONS</td>
<td>USAGE privilege on schema that contains the table</td>
</tr>
</tbody>
</table>

### Views

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE VIEW</td>
<td>• CREATE privilege on the schema to contain a view</td>
</tr>
<tr>
<td></td>
<td>• SELECT privileges on base objects (tables/views)</td>
</tr>
<tr>
<td></td>
<td>• USAGE privileges on schema that contains the base objects</td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>Must be the view owner and have USAGE privileges on the schema, or be the schema owner.</td>
</tr>
<tr>
<td>SELECT ... FROM VIEW</td>
<td>• SELECT privilege on view</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on the schema that contains the view.</td>
</tr>
<tr>
<td></td>
<td>• View owner must have SELECT ... WITH GRANT OPTION privileges on the view's anchor tables or views if non-owner runs a SELECT query on the view.</td>
</tr>
</tbody>
</table>
### Operation Required Privileges

- View owner must have SELECT privilege on a view's base objects (table or view) if owner runs a SELECT query on the view.

### Projections

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
</table>
| CREATE PROJECTION          | • SELECT privilege on anchor tables  
                              | • USAGE privilege on schema that contains anchor tables or schema owner  
                              | • CREATE privilege on schema to contain the projection  
                              | Note: If a projection is implicitly created with the table, no additional privilege is needed other than privileges for table creation. |

| AUTO/Delayed Projection    | On projections created during INSERT..SELECT or COPY operations:  
                              | • SELECT privilege on anchor tables  
                              | • USAGE privilege on schema that contains anchor tables |

| ALTER PROJECTION RENAME    | USAGE and CREATE privilege on schema that contains the projection |

| DROP PROJECTION            | USAGE privilege on schema that contains the projection or schema owner |

### External Procedures

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE PROCEDURE</td>
<td>Superuser</td>
</tr>
<tr>
<td>DROP PROCEDURE</td>
<td>Superuser</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>• EXECUTE privilege on procedure</td>
</tr>
</tbody>
</table>
### Libraries

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE LIBRARY</td>
<td>Superuser</td>
</tr>
<tr>
<td>DROP LIBRARY</td>
<td>Superuser</td>
</tr>
</tbody>
</table>

### User-Defined Functions

The following abbreviations are used in the UDF table:

- **UDF** = Scalar
- **UDT** = Transform
- **UDAnF** = Analytic
- **UDAF** = Aggregate

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE FUNCTION (SQL)</td>
<td>CREATE privilege on schema to contain the function</td>
</tr>
<tr>
<td>CREATE FUNCTION (UDF)</td>
<td>USAGE privilege on base library (if applicable)</td>
</tr>
<tr>
<td>CREATE TRANSFORM FUNCTION (UDF)</td>
<td></td>
</tr>
<tr>
<td>CREATE ANALYTIC FUNCTION (UDAnF)</td>
<td></td>
</tr>
<tr>
<td>CREATE AGGREGATE FUNCTION (UDAF)</td>
<td></td>
</tr>
<tr>
<td>DROP FUNCTION</td>
<td>Superuser or function owner</td>
</tr>
<tr>
<td>DROP TRANSFORM FUNCTION</td>
<td>USAGE privilege on schema that contains the function</td>
</tr>
<tr>
<td>DROP ANALYTIC FUNCTION</td>
<td></td>
</tr>
<tr>
<td>DROP AGGREGATE FUNCTION</td>
<td></td>
</tr>
<tr>
<td>ALTER FUNCTION (UDF) RENAME TO</td>
<td>USAGE and CREATE privilege on schema that contains the function</td>
</tr>
<tr>
<td>ALTER FUNCTION (UDF) SET SCHEMA</td>
<td>USAGE privilege on schema that currently</td>
</tr>
</tbody>
</table>
### Operation | Required Privileges
--- | ---
 | contains the function (old schema)
 | CREATE privilege on the schema to which the function will be moved (new schema)
| EXECUTE (SQL/UDF/UDT/ ADAF/UDAnF) function | EXECUTE privilege on function
 | USAGE privilege on schema that contains the function

### Sequences

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE SEQUENCE</td>
<td>CREATE privilege on schema to contain the sequence</td>
</tr>
<tr>
<td><strong>Note</strong>: Referencing sequence in the CREATE TABLE statement requires SELECT privilege on sequence object and USAGE privilege on sequence schema.</td>
<td></td>
</tr>
<tr>
<td>CREATE TABLE with SEQUENCE</td>
<td>SELECT privilege on sequence</td>
</tr>
<tr>
<td>USAGE privilege on sequence schema</td>
<td></td>
</tr>
<tr>
<td>DROP SEQUENCE</td>
<td>USAGE privilege on schema containing the sequence or schema owner</td>
</tr>
<tr>
<td>ALTER SEQUENCE RENAME TO</td>
<td>USAGE and CREATE privileges on schema</td>
</tr>
<tr>
<td>ALTER SEQUENCE SET SCHEMA</td>
<td>USAGE privilege on the schema that currently contains the sequence (old schema)</td>
</tr>
<tr>
<td>CREATE privilege on new schema to contain the sequence</td>
<td></td>
</tr>
<tr>
<td>CURRVAL()</td>
<td>SELECT privilege on sequence</td>
</tr>
<tr>
<td>NEXTVAL()</td>
<td>USAGE privilege on sequence schema</td>
</tr>
</tbody>
</table>
### Resource Pools

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE RESOURCE POOL</td>
<td>Superuser</td>
</tr>
<tr>
<td>ALTER RESOURCE POOL</td>
<td>Superuser on the resource pool to alter:</td>
</tr>
<tr>
<td></td>
<td>- MAXMEMORYSIZE</td>
</tr>
<tr>
<td></td>
<td>- PRIORITY</td>
</tr>
<tr>
<td></td>
<td>- QUEUETIMEOUT</td>
</tr>
<tr>
<td>UPDATE privilege on the resource pool to alter:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- PLANNEDCONCURRENCY</td>
</tr>
<tr>
<td></td>
<td>- SINGLEINITIATOR</td>
</tr>
<tr>
<td></td>
<td>- MAXCONCURRENCY</td>
</tr>
<tr>
<td>SET SESSION RESOURCE_POOL</td>
<td>- USAGE privilege on the resource pool</td>
</tr>
<tr>
<td></td>
<td>- Users can only change their own resource pool setting using ALTER USER syntax</td>
</tr>
<tr>
<td>DROP RESOURCE POOL</td>
<td>Superuser</td>
</tr>
</tbody>
</table>

### Users/Profiles/Roles

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE USER</td>
<td>Superuser</td>
</tr>
<tr>
<td>CREATE PROFILE</td>
<td></td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td></td>
</tr>
<tr>
<td>ALTER USER</td>
<td>Superuser</td>
</tr>
<tr>
<td>ALTER PROFILE</td>
<td></td>
</tr>
<tr>
<td>ALTER ROLE</td>
<td></td>
</tr>
<tr>
<td>ALTER ROLE RENAME</td>
<td></td>
</tr>
<tr>
<td>DROP USER</td>
<td>Superuser</td>
</tr>
<tr>
<td>DROP PROFILE</td>
<td></td>
</tr>
</tbody>
</table>
Object Visibility

You can use one or a combination of `vsql \d [pattern]` meta commands and SQL system tables to view objects on which you have privileges to view.

- Use `\dn [pattern]` to view schema names and owners
- Use `\dt [pattern]` to view all tables in the database, as well as the system table `V_CATALOG.TABLES`
- Use `\dj [pattern]` to view projections showing the schema, projection name, owner, and node, as well as the system table `V_CATALOG.PROJECTIONS`

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look up schema</td>
<td>At least one privilege on schema that contains the object</td>
</tr>
</tbody>
</table>
| Look up Object in Schema or in System Tables | USAGE privilege on schema  
At least one privilege on any of the following objects:  
TABLE  
VIEW  
FUNCTION  
PROCEDURE  
SEQUENCE |
<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look up Projection</td>
<td>At least one privilege on all anchor tables</td>
</tr>
<tr>
<td></td>
<td>USAGE privilege on schema of all anchor table</td>
</tr>
<tr>
<td>Look up resource pool</td>
<td>SELECT privilege on the resource pool</td>
</tr>
<tr>
<td>Existence of object</td>
<td>USAGE privilege on the schema that contains the object</td>
</tr>
</tbody>
</table>

### I/O Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTDISCONNECT</td>
<td>None</td>
</tr>
<tr>
<td>EXPORT TO Vertica</td>
<td>• SELECT privileges on the source table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on source table schema</td>
</tr>
<tr>
<td></td>
<td>• INSERT privileges for the destination table in target database</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on destination table schema</td>
</tr>
<tr>
<td>COPY FROM Vertica</td>
<td>• SELECT privileges on the source table</td>
</tr>
<tr>
<td></td>
<td>• USAGE privilege on source table schema</td>
</tr>
<tr>
<td></td>
<td>• INSERT privileges for the destination table in target database</td>
</tr>
</tbody>
</table>
### Operation Required Privileges

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
</table>
| COPY FROM file | database  
  - USAGE privilege on destination table schema |
| COPY FROM STDIN | Superuser  
  - INSERT privilege on table  
  - USAGE privilege on schema |
| COPY LOCAL |  
  - INSERT privilege on table  
  - USAGE privilege on schema |

### Comments

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
</table>
| COMMENT ON { is one of }: | Object owner or superuser  
  - AGGREGATE FUNCTION  
  - ANALYTIC FUNCTION  
  - COLUMN  
  - CONSTRAINT  
  - FUNCTION  
  - LIBRARY  
  - NODE  
  - PROJECTION  
  - SCHEMA  
  - SEQUENCE  
  - TABLE  
  - TRANSFORM FUNCTION |
### Operation Required Privileges

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW</td>
<td></td>
</tr>
</tbody>
</table>

### Transactions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMIT</td>
<td>None</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>None</td>
</tr>
<tr>
<td>RELEASE SAVEPOINT</td>
<td>None</td>
</tr>
<tr>
<td>SAVEPOINT</td>
<td>None</td>
</tr>
</tbody>
</table>

### Sessions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET { is one of }:</td>
<td></td>
</tr>
<tr>
<td>DATESTYLE</td>
<td>None</td>
</tr>
<tr>
<td>ESCAPE_STRING_WARNING</td>
<td></td>
</tr>
<tr>
<td>INTERVALSTYLE</td>
<td>None</td>
</tr>
<tr>
<td>LOCALE</td>
<td>None</td>
</tr>
<tr>
<td>ROLE</td>
<td>None</td>
</tr>
<tr>
<td>SEARCH_PATH</td>
<td>None</td>
</tr>
<tr>
<td>SESSION AUTOCOMMIT</td>
<td>None</td>
</tr>
<tr>
<td>SESSION CHARACTERISTICS</td>
<td></td>
</tr>
<tr>
<td>SESSION MEMORYCAP</td>
<td>None</td>
</tr>
<tr>
<td>SESSION RESOURCE POOL</td>
<td></td>
</tr>
<tr>
<td>SESSION RUNTIMECAP</td>
<td>None</td>
</tr>
<tr>
<td>Operation</td>
<td>Required Privileges</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>SESSION TEMPSPACE</td>
<td></td>
</tr>
<tr>
<td>STANDARD_CONFORMING_STRINGS</td>
<td></td>
</tr>
<tr>
<td>TIMEZONE</td>
<td></td>
</tr>
<tr>
<td>SHOW { name</td>
<td>ALL }</td>
</tr>
</tbody>
</table>

**Tuning Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE</td>
<td>Same privileges required to run the query being profiled</td>
</tr>
<tr>
<td>EXPLAIN</td>
<td>Same privileges required to run the query for which you use the EXPLAIN keyword</td>
</tr>
</tbody>
</table>
Privileges That Can Be Granted on Objects

The following table provides an overview of privileges that can be granted on (or revoked from) database objects in Vertica:

<table>
<thead>
<tr>
<th>Object</th>
<th>Create</th>
<th>Usage</th>
<th>Execute</th>
<th>Select</th>
<th>Insert</th>
<th>Delete</th>
<th>Update</th>
<th>References</th>
<th>Read</th>
<th>Write</th>
<th>Truncate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Pool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage Location</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

See Also

- [GRANT Statements](#)
- [REVOKE Statements](#)
Database Privileges

Only a database superuser can create a database. In a new database, the **PUBLIC Role** is granted USAGE on the automatically-created PUBLIC schema. It is up to the superuser to grant further privileges to users and roles.

The only privilege a superuser can grant on the database itself is CREATE, which allows the user to create a new schema in the database. For details on granting and revoking privileges on a database, see the **GRANT (Database)** and **REVOKE (Database)** topics in the SQL Reference Manual.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Grantor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Superuser</td>
<td>Allows a user to create a schema.</td>
</tr>
</tbody>
</table>

Schema Privileges

By default, only a superuser and the schema owner have privileges to create objects within a schema. Additionally, only the schema owner or a superuser can drop or alter a schema. See **DROP SCHEMA** and **ALTER SCHEMA**.

You must grant all new users access to the PUBLIC schema by running **GRANT USAGE ON SCHEMA PUBLIC**. Then grant new users CREATE privileges and privileges to individual objects in the schema. This enables new users to create or locate objects in the PUBLIC schema. Without USAGE privilege, objects in the schema cannot be used or altered, even by the object owner.

CREATE gives the schema owner or user WITH GRANT OPTION permission to create new objects in the schema, including renaming an object in the schema or moving an object into this schema.

**Note:** The schema owner is typically the user who creates the schema. However, a superuser can create a schema and assign ownership of the schema to a different user at creation.

All other access to the schema and its objects must be explicitly granted to users or roles by the superuser or schema owner. This prevents unauthorized users from accessing the schema and its objects. A user can be granted one of the following privileges through the **GRANT** statement:
<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Allows the user to create new objects within the schema. This includes the ability to create a new object, rename existing objects, and move objects into the schema from other schemas.</td>
</tr>
<tr>
<td>USAGE</td>
<td>Permission to select, access, alter, and drop objects in the schema. The user must also be granted access to the individual objects in order to alter them. For example, a user would need to be granted USAGE on the schema and SELECT on a table to be able to select data from a table. You receive an error message if you attempt to query a table that you have SELECT privileges on, but do not have USAGE privileges for the schema that contains the table.</td>
</tr>
</tbody>
</table>

Note the following on error messages related to granting privileges on a schema or an object:

- You attempt to grant a privilege to a schema, but you do not have USAGE privilege for the schema. In this case, you receive an error message that the schema does not exist.

- You attempt to grant a privilege to an object within a schema, and you have USAGE privilege on the schema. You do not have privilege on the individual object within the schema. In this case, you receive an error denying permission for that object.

Schema Privileges and the Search Path

The search path determines to which schema unqualified objects in SQL statements belong.

When a user specifies an object name in a statement without supplying the schema in which the object exists (called an unqualified object name) Vertica has two different behaviors, depending on whether the object is being accessed or created.

<table>
<thead>
<tr>
<th>Creating an object</th>
<th>Accessing/altering an object</th>
</tr>
</thead>
</table>
| When a user creates an object—such as table, view, sequence, procedure, function—with an unqualified name, Vertica tries to create the object in the current schema (the first schema in the schema search path), returning an error if the schema does not exist or if the user does not have CREATE privileges in that schema. Use the SHOW search_path command to view the current search path.  

```
=> SHOW search_path;  name | setting
-----------------------------------------
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```
| When a user accesses or alters an object with an unqualified name, those statements search through all schemas for a matching object, starting with the current schema, where:  
- The object name in the schema matches the object name in the |

Vertica Analytic Database (9.0.x)
Creating an object | Accessing/altering an object
--- | ---
**Note:** The first schema in the search path is the current schema, and the $user setting is a placeholder that resolves to the current user's name. | statement.
- The user has USAGE privileges on the schema in order to access object in it.
- The user has at least one privilege on the object.

See Also

- Setting Search Paths
- GRANT (Schema)
- REVOKE (Schema)

Table Privileges

By default, only a superuser and the table owner (typically the person who creates a table) have access to a table. The ability to drop or alter a table is also reserved for a superuser or table owner. This privilege cannot be granted to other users.

All other users or roles (including the user who owns the schema, if he or she does not also own the table) must be explicitly granted using WITH GRANT OPTION syntax to access the table.

These are the table privileges a superuser or table owner can grant:

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Permission to run SELECT queries on the table.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Permission to INSERT data into the table.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Permission to DELETE data from the table, as well as SELECT privilege on the table when executing a DELETE statement that references table column values in a WHERE or SET clause.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Permission to UPDATE and change data in the table, as well as SELECT privilege</td>
</tr>
</tbody>
</table>
### Privilege

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCES</td>
<td>Permission to CREATE foreign key constraints that reference this table.</td>
</tr>
</tbody>
</table>

on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause.

To use any of the above privileges, the user must also have USAGE privileges on the schema that contains the table. See Schema Privileges for details.

Referencing sequence in the CREATE TABLE statement requires the following privileges:

- SELECT privilege on sequence object
- USAGE privilege on sequence schema

For details on granting and revoking table privileges, see GRANT (Table) and REVOKE (Table) in the SQL Reference Manual.

### Projection Privileges

Because projections are the underlying storage construct for tables, they are atypical in that they do not have an owner or privileges associated with them directly. Instead, the privileges to create, access, or alter a projection are based on the anchor tables that the projection references, as well as the schemas that contain them.

All queries in Vertica obtain data from projections directly or indirectly. In both cases, to run a query, you must have SELECT privileges on the tables that the projections reference, and USAGE privileges on all schemas that contain those tables.

You can create projections in two ways: explicitly and implicitly.

### Explicit Projection Creation

To create a projection with CREATE PROJECTION or any of its variants, you must be a superuser or owner of the anchor table.

Only the anchor table owner can drop explicitly created projections. Explicitly created projections can be live aggregate projections, including Top-K projections and projections with expressions.
Implicit Projection Creation

When you insert data into a table, Vertica automatically creates a superprojection for the table.

Superprojections do not require any additional privileges to create or drop, other than privileges for table creation. Users who can create a table or drop a table can also create and drop the associated superprojection.

See Also

- ALTER PROJECTION RENAME
- CREATE PROJECTION
- DROP PROJECTION

View Privileges

A view is a stored query that dynamically accesses and computes data from the database at execution time. Use \dv in vsql to display available views. By default, only the following users have privileges to access a view's base object:

- Superuser
- View owner—typically, the view creator

To execute a query that contains a view, you must have:

- SELECT privileges assigned with GRANT (View)
- USAGE privileges on the view's schema, assigned with GRANT (Schema).

You can assign view privileges to other users and roles using GRANT (View). For example:

- Assign GRANT ALL privileges to a user or role.

=> GRANT all privileges on view1 to role1 with grant option;
• Assign GRANT ROLE privileges to a specific role to provide view privileges. In the following example, privileges that are assigned to role1 are assigned to role2:

```sql
=> CREATE ROLE role1;
=> CREATE ROLE role2;
=> GRANT role1 to role2;
```

See Also

**GRANT (View)**

**REVOKE (View)**

### Named Sequence Privileges

A named sequence can require up to three sets of privileges:

- **Create Privileges**
- **Usage Privileges**
- **Management Privileges**

#### Create Privileges

To create a sequence, a user must have CREATE privileges on a schema that contains the sequence.

#### Usage Privileges

Only the sequence creator/owner and superusers can initially access a new sequence. Other users must be granted the following privileges by a superuser or the sequence owner:

- SELECT privilege on the sequence.
- USAGE privilege on the schema that contains the sequence.

These privileges enable users to invoke functions **CURRVAL** and **NEXTVAL** on a sequence, either directly on the sequence itself, or implicitly through a table that references the sequence.
Management Privileges

Vertica provides the following SQL statements to manage partitions. Each requires its own set of privileges:

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Required Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP SEQUENCE</td>
<td>Sequence owner or schema owner</td>
</tr>
<tr>
<td>ALTER SEQUENCE ...RENAME TO</td>
<td>Sequence owner with USAGE and CREATE privileges on the schema of the sequence to rename.</td>
</tr>
<tr>
<td>ALTER SEQUENCE ...SET SCHEMA</td>
<td>Sequence owner with USAGE privilege on the current schema, and CREATE privilege on the destination schema.</td>
</tr>
<tr>
<td>ALTER SEQUENCE ...OWNER TO</td>
<td>Current sequence owner.</td>
</tr>
</tbody>
</table>

For details on granting and revoking sequence privileges, see GRANT (Sequence) and REVOKE (Sequence) in the SQL Reference Manual.

External Procedure Privileges

Only a superuser is allowed to create or drop an external procedure.

By default, users cannot execute external procedures. A superuser must grant users and roles this right, using the GRANT (Procedure) EXECUTE statement. Additionally, users must have USAGE privileges on the schema that contains the procedure in order to call it.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE</td>
<td>Permission to run an external procedure.</td>
</tr>
<tr>
<td>USAGE</td>
<td>Permission on the schema that contains the procedure.</td>
</tr>
</tbody>
</table>

For details on granting and revoking external table privileges, see GRANT (Procedure) and REVOKE (Procedure) in the SQL Reference Manual.
User-Defined Function Privileges

User-defined functions (described in CREATE FUNCTION Statements) can be created by superusers or users with CREATE privileges on the schema that will contain the function, as well as USAGE privileges on the base library (if applicable).

Users or roles other than the function owner can use a function only if they have been granted EXECUTE privileges on it. They must also have USAGE privileges on the schema that contains the function to be able to call it.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE</td>
<td>Permission to call a user-defined function.</td>
</tr>
<tr>
<td>USAGE</td>
<td>Permission on the schema that contains the function.</td>
</tr>
</tbody>
</table>

- **DROP FUNCTION**: Only a superuser or function owner can drop the function.
- **ALTER FUNCTION (UDF) RENAME TO**: A superuser or function owner must have USAGE and CREATE privileges on the schema that contains the function to be renamed.
- **ALTER FUNCTION (UDF) SET SCHEMA**: A superuser or function owner must have USAGE privilege on the schema that currently contains the function (old schema), as well as CREATE privilege on the schema where the function will be moved (new schema).

For details on granting and revoking user-defined function privileges, see the following topics in the SQL Reference Manual:

- **GRANT (User Defined Extension)**
- **REVOKE (User Defined Extension)**

Library Privileges

Only a superuser can load an external library using the CREATE LIBRARY statement. By default, only a superuser can create user-defined functions (UDFs) based on a loaded library. A superuser can use the GRANT USAGE ON LIBRARY statement to allow users to create UDFs based on classes in the library. The user must also have CREATE privileges on the schema that will contain the UDF.
Privilege | Description
---|---
USAGE | Permission to create UDFs based on classes in the library

Once created, only a superuser or the user who created a UDF can use it by default. Either of them can grant other users or roles the ability to call the function using the GRANT EXECUTE ON FUNCTION statement. See the GRANT (User Defined Extension) and REVOKE (User Defined Extension) topics in the SQL Reference Manual for more information on granting and revoking privileges on functions.

In addition to EXECUTE privilege, users/roles also require USAGE privilege on the schema in which the function resides in order to execute the function.

For more information about libraries and UDFs, see Developing User-Defined Extensions (UDxs) in Extending Vertica.

### Resource Pool Privileges

Only a superuser can create, alter, or drop a resource pool.

By default, users are granted USAGE rights to the GENERAL pool, from which their queries and other statements allocate memory and get their priorities. A superuser must grant users USAGE rights to any additional resource pools by using the GRANT USAGE ON RESOURCE POOL statement. Once granted access to the resource pool, users can use the SET SESSION RESOURCE_POOL statement and the RESOURCE POOL clause of the ALTER USER statement to have their queries draw their resources from the new pool.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USAGE</td>
<td>Permission to use a resource pool.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Permission to look up resource pool information/status in system tables.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Permission to adjust the tuning parameters of the pool.</td>
</tr>
</tbody>
</table>

For details on granting and revoking resource pool privileges, see GRANT (Resource Pool) and REVOKE (Resource Pool) in the SQL Reference Manual.

### Storage Location Privileges

Users and roles without superuser privileges can copy data to and from storage locations as long as the following conditions are met, where a superuser:

Vertica Analytic Database (9.0.x)
1. Creates a special class of storage location (CREATE LOCATION) specifying the USAGE argument set to 'USER', which indicates the specified area is accessible to non-superusers users.

2. Grants users or roles READ and/or WRITE access to the specified location using the GRANT (Storage Location) statement.

    **Note:** GRANT/REVOKE (Storage Location) statements are applicable only to 'USER' storage locations.

Once such storage locations exist and the appropriate privileges are granted, users and roles granted READ privileges can copy data from files in the storage location into a table. Those granted WRITE privileges can export data from a table to the storage location on which they have been granted access. WRITE privileges also let users save COPY statement exceptions and rejected data files from Vertica to the specified storage location.

Only a superuser can add, alter, retire, drop, and restore a location, as well as set and measure location performance. All non-dbadmin users or roles require READ and/or WRITE permissions on the location.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>Allows the user to copy data from files in the storage location into a table.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Allows the user to copy data to the specific storage location. Users with WRITE privileges can also save COPY statement exceptions and rejected data files to the specified storage location.</td>
</tr>
</tbody>
</table>

See Also

- GRANT (Storage Location)
- Storage Management Functions
- CREATE LOCATION

Role, profile, and User Privileges

Only a superuser can create, alter or drop a:
By default, only the superuser can grant or revoke a role to another user or role. A user or role can be given the privilege to grant and revoke a role by using the WITH ADMIN OPTION clause of the GRANT statement.

For details on granting and revoking role privileges, see GRANT (Role) and REVOKE (Role) in the SQL Reference Manual.

See Also

- CREATE USER
- ALTER USER
- DROP USER
- CREATE PROFILE
- ALTER PROFILE
- DROP PROFILE
- CREATE ROLE
- ALTER ROLE RENAME
- DROP ROLE

**Metadata Privileges**

A superuser has unrestricted access to all database metadata. Other users have significantly reduced access to metadata based on their privileges, as follows:

<table>
<thead>
<tr>
<th>Type of Metadata</th>
<th>User Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog objects:</td>
<td>Users must possess USAGE privilege on the schema and any type of access (SELECT) or modify privilege on the</td>
</tr>
<tr>
<td>Tables</td>
<td></td>
</tr>
<tr>
<td>Type of Metadata</td>
<td>User Access</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Columns</td>
<td>object to see catalog metadata about the object. See also Schema Privileges.</td>
</tr>
<tr>
<td>Constraints</td>
<td>For internal objects like projections, WOS and ROS containers that don't have access privileges directly associated with them, the user must possess the requisite privileges on the associated schema and table objects instead. For example, to see whether a table has any data in the WOS, you need to have USAGE on the table schema and at least SELECT on the table itself. See also Table Privileges and Projection Privileges.</td>
</tr>
<tr>
<td>Sequences</td>
<td></td>
</tr>
<tr>
<td>External Procedures</td>
<td></td>
</tr>
<tr>
<td>Projections</td>
<td></td>
</tr>
<tr>
<td>ROS containers</td>
<td></td>
</tr>
<tr>
<td>WOS</td>
<td></td>
</tr>
</tbody>
</table>

User sessions and functions, and system tables related to these sessions

Users can only access information about their own, current sessions.

The following functions provide restricted functionality to users:

- `CURRENT_DATABASE`
- `CURRENT_SCHEMA`
- `CURRENT_USER`
- `HAS_TABLE_PRIVILEGE`
- `SESSION_USER` (same as `CURRENT_USER`)

The system table, `SESSIONS`, provides restricted functionality to users.

Storage locations

Users require READ permissions to copy data from storage locations.

Only a superuser can add or retire storage locations.
I/O Privileges

Users need no special permissions to connect to and disconnect from a Vertica database.

To **EXPORT TO** and **COPY FROM** Vertica, the user must have:

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

To **COPY FROM** STDIN and use local **COPY** a user must have INSERT privileges on the table and USAGE privilege on schema.

**Note:** Only a superuser can **COPY from file**.

Comment Privileges

A comment lets you add, revise, or remove a textual message to a database object. You must be an object owner or superuser in order to COMMENT ON one of the following objects:

- **COLUMN**
- **CONSTRAINT**
- **FUNCTION** (including AGGREGATE and ANALYTIC)
- **LIBRARY**
- **NODE**
- **PROJECTION**
- **SCHEMA**
- **SEQUENCE**
- **TABLE**
- **TRANSFORM FUNCTION**
- **VIEW**

Other users must have VIEW privileges on an object to view its comments.

**Transaction Privileges**

No special permissions are required for the following database operations:

- **COMMIT**
- **ROLLBACK**
- **RELEASE SAVEPOINT**
- **SAVEPOINT**

**Session Privileges**

No special permissions are required for users to use the SHOW statement or any of the SET statements.

**Tuning Privileges**

In order to **PROFILE** a single SQL statement or returns a query plan's execution strategy to standard output using the **EXPLAIN** command, users must have the same privileges that are required for them to run the same query without the PROFILE or EXPLAIN keyword.

**Granting and Revoking Privileges**

To grant or revoke a privilege using one of the SQL GRANT or REVOKE statements, the user must have the following permissions for the GRANT/REVOKE statement to succeed:

- Superuser or privilege WITH GRANT OPTION
- **USAGE** privilege on the schema
- Appropriate privileges on the object
The syntax for granting and revoking privileges is different for each database object, such as schema, database, table, view, sequence, procedure, function, resource pool, and so on.

Normally, a superuser first creates a user and then uses GRANT syntax to define the user's privileges or roles or both. For example, the following series of statements creates user Carol and grants Carol access to the apps database in the PUBLIC schema and also lets Carol grant SELECT privileges to other users on the applog table:

```
=> CREATE USER Carol;
=> GRANT USAGE ON SCHEMA PUBLIC to Carol;
=> GRANT ALL ON DATABASE apps TO Carol;
=> GRANT SELECT ON applog TO Carol WITH GRANT OPTION;
```


About Superuser Privileges

A superuser (DBADMIN) is the automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization, as well as supersede any user that has been granted the PSEUDOSUPERUSER role.

Note: Database superusers are not the same as a Linux superuser with (root) privilege and cannot have Linux superuser privilege.

A superuser can grant privileges on all database object types to other users, as well as grant privileges to roles. Users who have been granted the role will then gain the privilege as soon as they enable it.

Superusers may grant or revoke any object privilege on behalf of the object owner, which means a superuser can grant or revoke the object privilege if the object owner could have granted or revoked the same object privilege. A superuser may revoke the privilege that an object owner granted, as well as the reverse.

Since a superuser is acting on behalf of the object owner, the GRANTOR column of V_CATALOG.GRANTS table displays the object owner rather than the superuser who issued the GRANT statement.

A superuser can also alter ownership of table and sequence objects.

See Also

DBADMIN Role
About Schema Owner Privileges

By default, the schema owner has privileges to create objects within a schema. Additionally, the schema owner can drop any object in the schema, requiring no additional privilege on the object.

The schema owner is typically the user who creates the schema.

Schema owners cannot access objects in the schema. Access to objects requires the appropriate privilege at the object level.

All other access to the schema and its objects must be explicitly granted to users or roles by a superuser or schema owner to prevent unauthorized users from accessing the schema and its objects.

See Schema Privileges

About Object Owner Privileges

The database, along with every object in it, has an owner. The object owner is usually the person who created the object, although a superuser can alter ownership of objects, such as table and sequence.

Object owners must have appropriate schema privilege to access, alter, rename, move or drop any object it owns without any additional privileges.

An object owner can also:

- Grant privileges on their own object to other users
  
  The WITH GRANT OPTION clause specifies that a user can grant the permission to other users. For example, if user Bob creates a table, Bob can grant privileges on that table to users Ted, Alice, and so on.

- Grant privileges to roles
  
  Users who are granted the role gain the privilege.

How to Grant Privileges

As described in Granting and Revoking Privileges, specific users grant privileges using the GRANT statement with or without the optional WITH GRANT OPTION, which allows the user to grant the same privileges to other users.
The user needs to have USAGE privilege on schema and appropriate privileges on the object.

When a user grants an explicit list of privileges, such as

```sql
GRANT INSERT,DELETE,REFERENCES ON applog TO Bob;
```

- The GRANT statement succeeds only if all the roles are granted successfully. If any grant operation fails, the entire statement rolls back.
- Vertica will return ERROR if the user does not have grant options for the privileges listed.

When a user grants ALL privileges, such as

```sql
GRANT ALL ON applog TO Bob,
```

the statement always succeeds. Vertica grants all the privileges on which the grantor has the WITH GRANT OPTION and skips those privileges without the optional WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only DELETE privileges are granted to Bob, and the statement succeeds:

```sql
=> GRANT DELETE ON applog TO Bob WITH GRANT OPTION;GRANT PRIVILEGE
```

For details, see the [GRANT Statements](#) in the SQL Reference Manual.

## How to Revoke Privileges

In general, ONLY the user who originally granted a privilege can revoke it using a REVOKE statement. That user must have superuser privilege or have the optional WITH GRANT OPTION on the privilege. The user also must have USAGE privilege on the schema and appropriate privileges on the object for the REVOKE statement to succeed.

In order to revoke a privilege, this privilege must have been granted to the specified grantee by this grantor before. If Vertica finds that to be the case, the above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if supplied) from the grantee. Otherwise, Vertica prints a NOTICE that the operation failed, as in the following example.

```sql
=> REVOKE SELECT ON applog FROM Bob;
NOTICE @: Cannot revoke "SELECT" privilege(s) for relation "applog" that you did not grant to "Bob"
```
The above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if applicable) from the grantee or it prints a notice that the operation failed.

In order to revoke grant option for a privilege, the grantor must have previously granted the grant option for the privilege to the specified grantee. Otherwise, Vertica prints a NOTICE.

The following REVOKE statement removes the GRANT option only but leaves the privilege intact:

```
=> GRANT INSERT on applog TO Bob WITH GRANT OPTION;
GRANT PRIVILEGE
=> REVOKE GRANT OPTION FOR INSERT ON applog FROM Bob;
REVOKE PRIVILEGE
```

When a user revokes an explicit list of privileges, such as GRANT INSERT, DELETE, REFERENCES ON applog TO Bob:

- The REVOKE statement succeeds only if all the roles are revoked successfully. If any revoke operation fails, the entire statement rolls back.
- Vertica returns ERROR if the user does not have grant options for the privileges listed.
- Vertica returns NOTICE when revoking privileges that this user had not been previously granted.

When a user revokes ALL privileges, such as REVOKE ALL ON applog TO Bob, the statement always succeeds. Vertica revokes all the privileges on which the grantor has the optional WITH GRANT OPTION and skips those privileges without the WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only grant option is revoked from Bob, and the statement succeeds without NOTICE:

```
=> REVOKE GRANT OPTION FOR DELETE ON applog FROM Bob;
```

For details, see the REVOKE Statements in the SQL Reference Manual.

Privilege Ownership Chains

The ability to revoke privileges on objects can cascade throughout an organization. If the grant option was revoked from a user, the privilege that this user granted to other users will also be revoked.

If a privilege was granted to a user or role by multiple grantors, to completely revoke this privilege from the grantee the privilege has to be revoked by each original grantor. The only
exception is a superuser may revoke privileges granted by an object owner, with the reverse being true, as well.

In the following example, the SELECT privilege on table t1 is granted through a chain of users, from a superuser through User3.

- A superuser grants User1 CREATE privileges on the schema s1:

```sql
=> \c - dbadmin
You are now connected as user "dbadmin".
=> CREATE USER User1;
CREATE USER
=> CREATE USER User2;
CREATE USER
=> CREATE USER User3;
CREATE USER
=> CREATE SCHEMA s1;
CREATE SCHEMA
=> GRANT USAGE on SCHEMA s1 TO User1, User2, User3;
GRANT PRIVILEGE
=> CREATE ROLE reviewer;
CREATE ROLE
=> GRANT CREATE ON SCHEMA s1 TO User1;
GRANT PRIVILEGE
```

- User1 creates new table t1 within schema s1 and then grants SELECT WITH GRANT OPTION privilege on s1.t1 to User2:

```sql
=> \c - User1
You are now connected as user "User1".
=> CREATE TABLE s1.t1(id int, sourceID VARCHAR(8));
CREATE TABLE
=> GRANT SELECT on s1.t1 to User2 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User2 grants SELECT WITH GRANT OPTION privilege on s1.t1 to User3:

```sql
=> \c - User2
You are now connected as user "User2".
=> GRANT SELECT on s1.t1 to User3 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User3 grants SELECT privilege on s1.t1 to the reviewer role:

```sql
=> \c - User3
You are now connected as user "User3".
=> GRANT SELECT on s1.t1 to reviewer;
GRANT PRIVILEGE
```

Users cannot revoke privileges upstream in the chain. For example, User2 did not grant privileges on User1, so when User1 runs the following REVOKE command, Vertica rolls back the command:
Users can revoke privileges indirectly from users who received privileges through a cascading chain, like the one shown in the example above. Here, users can use the CASCADE option to revoke privileges from all users "downstream" in the chain. A superuser or User1 can use the CASCADE option to revoke the SELECT privilege on table s1.t1 from all users. For example, a superuser or User1 can execute the following statement to revoke the SELECT privilege from all users and roles within the chain:

```sql
REVOKE SELECT ON s1.t1 FROM User2 CASCADE;
```
Modifying Privileges

A superuser or object owner can use one of the ALTER statements to modify a privilege, such as changing a sequence owner or table owner. Reassignment to the new owner does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Viewing Privileges Granted on Objects

Vertica logs information about privileges granted on various objects, including the grantor and grantee, in the V_CATALOG.GRANTS system table. The order of columns in the table corresponds to the order in which they appear in the GRANT command. An asterisk in the output means the privilege was granted WITH GRANT OPTION.

The following command queries the GRANTS system table:

```sql
=> SELECT * FROM grants ORDER BY grantor, grantee;

<table>
<thead>
<tr>
<th>grantor</th>
<th>privileges_description</th>
<th>object_schema</th>
<th>object_name</th>
<th>grantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>CREATE</td>
<td></td>
<td></td>
<td>Alice</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE</td>
<td></td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE</td>
<td></td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>dbadmin</td>
<td>INSERT, UPDATE, DELETE, REFERENCES</td>
<td>public</td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Ted</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Sue</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE, CREATE TEMP</td>
<td></td>
<td></td>
<td>Sue</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Sue</td>
</tr>
<tr>
<td>dbadmin</td>
<td>SELECT*</td>
<td>public</td>
<td></td>
<td>Sue</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Alice</td>
</tr>
<tr>
<td>dbadmin</td>
<td>INSERT, SELECT</td>
<td>public</td>
<td></td>
<td>commentor</td>
</tr>
<tr>
<td>dbadmin</td>
<td>INSERT, SELECT</td>
<td>public</td>
<td></td>
<td>commentor</td>
</tr>
<tr>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
<td>logwriter</td>
</tr>
<tr>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
<td>logadmin</td>
</tr>
<tr>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
<td>logreader</td>
</tr>
<tr>
<td>dbadmin</td>
<td>DELETE</td>
<td>public</td>
<td></td>
<td>logadmin</td>
</tr>
<tr>
<td>dbadmin</td>
<td>SELECT</td>
<td>public</td>
<td></td>
<td>logadmin</td>
</tr>
<tr>
<td>dbadmin</td>
<td>INSERT</td>
<td>public</td>
<td></td>
<td>logwriter</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>v_internal</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE TEMP</td>
<td></td>
<td></td>
<td>vmart</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>public</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>v_catalog</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>v_monitor</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE*, CREATE TEMP*</td>
<td></td>
<td></td>
<td>vmart</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE*, CREATE*</td>
<td></td>
<td></td>
<td>dbadmin</td>
</tr>
<tr>
<td>dbadmin</td>
<td>INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES*</td>
<td>public</td>
<td></td>
<td>dbadmin</td>
</tr>
</tbody>
</table>
```
To quickly find all of the privileges that have been granted to all users on the schema named myschema, run the following statement:

```sql
=> SELECT granteep | privilegeset FROM GRANTS WHERE object_name='myschema';
```

Note that the vsql commands, \dp and \z, both return similar information to GRANTS:

```sql
=> \dp
```


## Access Policies

You can create the following access policy types to restrict access to sensitive information to only those users authorized to view it:

- **Column Access Policy**
- **Row Access Policy**
Important: If you have a table with both a row level access policy and a column level access policy, Vertica filters the row level access policy first. Then Vertica uses the column level access policy to filter the columns.

Use Cases

Column Access Policy Use Case
Base a column access policy on a user's role and the privileges granted to that role.

For example, in a healthcare organization, customer support representatives and account managers have access to the same customer table. The table contains the column SSN for storing customer Social Security numbers. to which customer support representatives have only partial access, to view the last four digits. The account manager, however, must be able to view entire Social Security numbers. Therefore, the manager role has privileges to view all nine digits of the social security numbers.

When creating a column access policy, use expressions to specify exactly what different users or roles can access within the column.

In this case, a manager can access the entire SSN column, while customer support representatives can only access the last four digits:

```sql
=> CREATE ACCESS POLICY ON schema.customers_table
   FOR COLUMN SSN
   CASE
       WHEN ENABLED_ROLE('manager') THEN SSN
   else substr(SSN, 8, 4)
   END
   ENABLE;
```

Row Access Policy Use Case
You can also create a row access policy on the same table. For example, you can modify access to a customer table so a manager can view data in all rows. However, a broker can see a row only if the customer is associated with that broker:

```sql
=> select * from customers_table;

<table>
<thead>
<tr>
<th>custID</th>
<th>password</th>
<th>ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>secret</td>
<td>12345678901</td>
</tr>
<tr>
<td>2</td>
<td>secret</td>
<td>12345678902</td>
</tr>
<tr>
<td>3</td>
<td>secret</td>
<td>12345678903</td>
</tr>
</tbody>
</table>

(3 rows)
```
Each customer in the customers_table has an assigned broker:

```
=> select * from broker_info;
broker | custID
-------|--------
  u1    |    1
  u2    |    2
  u3    |    3
```

Create the access policy to allow a *manager* to see all data in all rows. Limit a broker's view to only those customers to which the broker is assigned:

```
=> CREATE ACCESS POLICY ON schema.customers_table
   FOR rows
   WHERE
     ENABLED_ROLE('manager')
   or
     (ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_info
                                                                 WHERE broker = CURRENT_USER()))
   ENABLE;
```

**Access Policy Creation Workflow**

You can create access policies for any table type, columnar, external, or flex. You can also create access policies on any column type, including joins.

If no users or roles are already created, you must create them before creating an access policy:

- Create a User
- Create a Role
- GRANT (Schema)
- GRANT (Table)
- Grant a user access to the role
- The user enables the role with the SET ROLE statement (unless the administration user assigned a default role to the user)
- Create the access policy with the CREATE ACCESS POLICY statement.
Working With Access Policies

This section describes areas that may affect how you use access policies.

Performing Operations

Having row and column access policies enabled on a table may affect the behavior when you attempt to perform the following DML operations:

- Insert
- Update
- Delete
- Merge
- Copy
- Select

Row Level Access Behavior

On tables where a row access policy is enabled, you can only perform DML operations when the condition in the Row access policy evaluates to TRUE. For example:

Table1 appears as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Create the following row access policy on Table1:

```sql
=> CREATE ACCESS POLICY on table1 for ROWS
    WHERE enabled_role('manager')
    OR
    A<2
    ENABLE;
```

With this policy enabled, the following behavior exists for users who want to perform DML operations:
A user with the manager role can perform DML on all rows in the table, because the WHERE clause in the policy evaluates to TRUE.

Users with non-manager roles can only perform a SELECT to return data in column A that has a value of less than two. If the access policy has to read the data in the table to confirm a condition, it does not allow DML operations.

**Column Level Access Behavior**

On tables where a column access policy is enabled, you can perform DML operations if you can view the entire column. For example:

Table1 appears as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Create the following column access policy on Table1:

```sql
=> CREATE ACCESS POLICY on Table1 FOR column A NULL::int enable;
```

In this case users cannot perform DML operations on column A.

**Important:** Users who can access all the rows and columns in a table with an access policy enabled can perform DML operations. Therefore, when you create an access policy, make sure you construct it in a manner that all row and column data is accessible by at least one user. This allows at least one user to perform any DML that may be required. Otherwise, you can temporarily disable the access policy to perform DML.

**Schema Table and Privileges**

Only dbadmin users can create access policies. If you want a user to be able to use access policies, you must first assign that user the appropriate privileges.

- Grant `schema` or `table` privileges to a table non-owner to allow that user to use the access policy.
- Revoke `schema` or `table` privileges to prohibit the user from using the access policy.
This example shows how you can create an access policy without the user being granted privilege for the public schema:

```sql
=> CREATE ACCESS POLICY ON public.customers_table
   FOR COLUMN SSN
   WHEN ENABLED_ROLE('operator') THEN SUBSTR(SSN, 8, 4)
```

Enable and Disable Access Policy Creation

Access policies are enabled by default for all tables in the database. To disable and enable the creation of new access policies at the database level, use the ALTER DATABASE statement.

Disable Creation of New Access Policies

```sql
=> ALTER DATABASE dbname SET EnableAccessPolicy=0;
```

Enable Creation of New Access Policies

```sql
=> ALTER DATABASE dbname SET EnableAccessPolicy=1;
```

Limitations on Creating Access Policies with Projections

You can create access policies on columns in tables that are part of a projection. However, you cannot create an access policy on an input table for the following projections:

- Top-K projections
- Aggregate projections
- Projections with expressions

Sometimes, a table already has an access policy and is part of a projection. In such cases, if the Vertica optimizer cannot fold (or compress) the query, the access query is blocked.

Query Optimization Considerations

When using access policies be aware of the following potential behaviors, and design tables optimally.

**Design Tables That All Authorized Users Can Access**

When Database Designer creates projections for a given table, it takes into account the access policies that apply to the current user. The set of projections that Database Designer produces for the table are optimized for that user's access privileges, and other users with similar access privileges. However, these projections might be less than optimal for users with different
access privileges. These differences might have some effect on how efficiently Vertica processes queries from those users. Therefore, when you evaluate projection designs for that table using Database Designer, design a given table so that all authorized users have optimal access.

**Avoid Performance Issues Caused by Dynamic Rewrite**

To enforce row-level access policies, the system dynamically rewrites user queries. Therefore, query performance may be affected by how row-level access policies are written.

For example, referring to preceding Access Policy Use Cases, run the following query. Enable both the row and column access policies on the customers_table:

```sql
=> SELECT * from customers_table;
```

Vertica rewrites this query plan to:

```sql
=> SELECT * from (select custID, password, CASE WHEN enabled_role('manager') THEN SSN else substr(SSN, 8, 4) end AS SSN
FROM customers_table
WHERE
enabled_role('broker')
AND
customers_table.custID IN (SELECT broker_info.custID FROM broker_info WHERE broker = current_user())
) customers_table;
```

**Column Access Policy**

Use the `CREATE ACCESS POLICY` statement to create a column access policy for a specific column in a table. Creating an access policy depends on the expressions specified when creating the policy, and also on the following:

- **Viewing a User's Role**
- **Granting Privileges to Roles**

**Example**

Suppose you want to prevent users from viewing a specific column in a table. This example shows how to create an access policy that masks column A in Table1.

Run the following SQL command:
=> SELECT * FROM Table1;

Table1 appears as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one</td>
</tr>
<tr>
<td>2</td>
<td>two</td>
</tr>
<tr>
<td>3</td>
<td>three</td>
</tr>
<tr>
<td>4</td>
<td>four</td>
</tr>
</tbody>
</table>

Create the following column access policy:

=> CREATE ACCESS POLICY on Table1 FOR column A NULL::int enable;

Re-run the SQL command:

=> SELECT * FROM Table1;

The following is returned:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one</td>
</tr>
<tr>
<td></td>
<td>two</td>
</tr>
<tr>
<td></td>
<td>three</td>
</tr>
<tr>
<td></td>
<td>four</td>
</tr>
</tbody>
</table>

Note that no values appear in column A because the access policy prevents the return of this data (NULL::int).

Creating Column Access Policies

Creating a column access policy allows different users to run the same query and receive different results. For example, you can create an access policy authorizing access to a column of bank account numbers. You can specify that a user with the role employee cannot access this information. However, you do give access to a user with a manager role.

Conditions specified in the access policy determine whether the user can see data restricted by the policy. This example shows how you can specify that the manager role can view the entire Social Security number while the operator role can only view the last four digits. The first five digits are masked for the operator role (THEN SUBSTR (SSN, 8, 4)). The 8 indicates the operator sees data starting on the eighth character (such as 123-45-6789).

=> CREATE ACCESS POLICY ON customers_table FOR COLUMN SSN
CASE
  WHEN ENABLED_ROLE('manager') THEN SSN
  WHEN ENABLED_ROLE('operator') THEN SUBSTR(SSN, 8, 4)
  ELSE NULL
END
ENABLE;
Access Policy Limitations

When you use column access policies, be aware of the following limitations:

- When using an access policy you cannot use any of the following in an expression:
  - Aggregate functions
  - Subquery
  - Analytics
  - UDT

- If the query cannot be folded by the Vertica optimizer, all functions other than SELECT are blocked. The following error message appears:

  ERROR 0: Unable to INSERT: "Access denied due to active access policy on table <tablename> for column <columnname>

  Note: Folding a query refers to the act of replacing deterministic expressions involving only constants, with their computed values.

- You cannot create a column access policy on temporary tables.

- It is recommended to not use a column access policy on a flex table. If you create a column access policy on a flex table, the following appears:

  WARNING 0: Column Access Policies on flex tables may not be completely secure
Examples

The following examples show how to create a column access policy for various situations.

Create Access Policy in Public Schema for Column in Customer Table

=> CREATE ACCESS POLICY on public.customer FOR COLUMN cid length('xxxx') enable;

Use Expression to Further Specify Data Access and Restrictions

In this example, a user with a supervisor role can see data from the deal_size column in the vendor_dimension table. However, a user assigned an employee role cannot.

=> CREATE ACCESS POLICY ON vendor_dimension FOR COLUMN deal_size
   CASE
   WHEN ENABLED_ROLE('supervisor') THEN deal_size
   WHEN ENABLED_ROLE('employee') THEN NULL
   END
   ENABLE;

Substitute Specific Data for Actual Data in Column

In this example, the value 1000 appears rather than the actual column data:

=> CREATE ACCESS POLICY on public.customer FOR COLUMN cid 1000 enable;
=> SELECT * FROM customer;
cid  | dist_code
------
1000  | 2
1000  | 10
(2 rows)
See Also

- CREATE ACCESS POLICY
- ALTER ACCESS POLICY
- DROP ACCESS POLICY

Enable or Disable Column Access Policy

If you have dbadmin privileges, you can enable and disable an individual access policy in a table, as the following examples show.

Enable Column Access Policy

```sql
=> ALTER ACCESS POLICY on customer FOR column customer_key enable;
```

Disable Column Access Policy

```sql
=> ALTER ACCESS POLICY on customer FOR column customer_key disable;
```

Row Access Policy

Use the CREATE ACCESS POLICY statement to create a row access policy for a specific row in a table. You must use a WHERE clause to set the access policy's condition.

Example

Suppose you have a customers table but want to limit users with the broker role to being able to view only customers to which they are assigned. This example shows how to create a row access policy that allows managers to view everything and brokers to view only their customers.

Run the following SQL statement:

```sql
=> SELECT * FROM customers_table;
```

The customers_table appears as follows:

<table>
<thead>
<tr>
<th>custID</th>
<th>password</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>secret</td>
<td>123456789</td>
</tr>
</tbody>
</table>
Run the following SQL statement:

```sql
=> SELECT * FROM broker_info;
```

The broker_info table shows that each customer has an assigned broker:

<table>
<thead>
<tr>
<th>broker</th>
<th>custID</th>
</tr>
</thead>
<tbody>
<tr>
<td>user1</td>
<td>1</td>
</tr>
<tr>
<td>user2</td>
<td>2</td>
</tr>
<tr>
<td>user3</td>
<td>3</td>
</tr>
</tbody>
</table>

Create the following access policy that only allows brokers to see customers to which they are associated:

```sql
=> CREATE ACCESS POLICY on customers_table for rows
WHERE
   ENABLED_ROLE('manager')
or
   (ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_info
   WHERE broker = CURRENT_USER()))
ENABLE;
```

As user1, run the following SQL command:

```
user1=> SELECT * FROM customers_table;
```

The following is returned because user1 is associated with custID 1:

<table>
<thead>
<tr>
<th>custID</th>
<th>password</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>secret</td>
<td>123456789</td>
</tr>
</tbody>
</table>

Creating Row Access Policies

Creating a row access policy determines what rows a user can access during a query. Row access policies include a WHERE clause that prompts the query to return only those rows where the condition is true. For example, a user with a BROKER role should only be able to
access customer information for which the user is a broker. You can write a predicate for this situation as follows:

```
WHERE ENABLED_ROLE('broker') AND customers_table.custID in (SELECT broker_info.custID FROM broker_info WHERE broker = CURRENT_USER())
```

You can use a row access policy to enforce this restriction. The following example shows how you can create a row access policy. This policy limits a user with a broker role to access information for customers whose custID in the customers_table matches the custID in the broker_info table.

```
=> CREATE ACCESS POLICY on customers_table
   for rows
   WHERE
       ENABLED_ROLE('broker')
       AND
       customers_table.custID in (SELECT broker_info.custID FROM broker_info WHERE broker = CURRENT_USER())
enable;
```
Row Access Policy Limitations

Be aware of the following limitations when using row access policies:

- You can only have one row access policy per table. If you need to add more later, place the policies in a single WHERE predicate and use `ALTER ACCESS POLICY` to enable the new condition.

- You cannot use row access policies on:
  - Tables with aggregate projections
  - Temporary tables
  - System tables. If you try to create a row access policy on a system table the following message appears:

    ```
    => ROLLBACK 0: Access policy cannot be created on system table <system table name>
    ```
  - Views

- When a row access policy exists on a table, you cannot create directed queries on that table.
Examples

The following examples show you can create a row access policy:

Create Access Policy in for specific row in Customer Table

=> CREATE ACCESS POLICY on customer FOR ROWS where cust_id > 3 enable;
See Also

- CREATE ACCESS POLICY
- ALTER ACCESS POLICY
- DROP ACCESS POLICY

Enable or Disable Row Access Policy

If you have dbadmin privileges, you can enabled and disable individual row access policies in a table, as the following examples show:

**Enable Row Access Policy**

=> ALTER ACCESS POLICY on customer FOR rows enable;

**Disable Row Access Policy**

=> ALTER ACCESS POLICY on customer FOR rows disable;
About Database Roles

To make managing permissions easier, use roles. A role is a collection of privileges that a superuser can grant to (or revoke from) one or more users or other roles. Using roles avoids having to manually grant sets of privileges user by user. For example, several users might be assigned to the administrator role. You can grant or revoke privileges to or from the administrator role, and all users with access to that role are affected by the change.

Note: Users must first enable a role before they gain all of the privileges that have been granted to it. See Enabling Roles.

Role Hierarchies

You can also use roles to build hierarchies of roles; for example, you can create an administrator role that has privileges granted non-administrator roles as well as to the privileges granted directly to the administrator role. See also Role Hierarchy.

Roles do no supersede manually-granted privileges, so privileges directly assigned to a user are not altered by roles. Roles just give additional privileges to the user.

Creating and Using a Role

Using a role follows this general flow:

1. A superuser creates a role using the CREATE ROLE statement.

2. A superuser or object owner grants privileges to the role using one of the GRANT statements.

3. A superuser or users with administrator access to the role grant users and other roles access to the role.

4. Users granted access to the role use the SET ROLE command to enable that role and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.
Tip: You can query the V_CATALOG system tables ROLES, GRANTS, and USERS to see any directly-assigned roles; however, these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). See the HAS_ROLE() function for additional information.

Roles on Management Console

When users sign in to the Management Console (MC), what they can view or do is governed by MC roles. For details, see About MC Users and About MC Privileges and Roles.
Types of Database Roles

Vertica has the following pre-defined roles:

- PUBLIC
- PSEUDOSUPERUSER
- DBADMIN
- DBDUSER
- SYSMONITOR

You cannot drop or rename predefined roles, however, you can grant to, or revoke from, predefined roles except to/from PUBLIC. You can also grant predefined roles to other roles and users.

Individual privileges may be granted to/revoked from predefined roles. See the SQL Reference Manual for all of the GRANT and REVOKE statements.

DBADMIN Role

Every database has the special DBADMIN role. A superuser (or someone with the PSEUDOSUPERUSER Role) can grant this role to or revoke this role from any user or role.

Users who enable the DBADMIN role gain these privileges:

- Create or drop users
- Create or drop schemas
- Create or drop roles
- Grant roles to other users
- View all system tables
- View and terminate user sessions
- Access to all data created by any user

The DBADMIN role does NOT allow users to:
- Start and stop a database
- Set configuration parameters

**Note:** A user with a DBADMIN role must have the ADMIN OPTION enabled to be able to grant a DBADMIN or PSEUDOSUPERUSER role to another user. For more information see [GRANT (Role)](#).

You cannot assign additional roles to the DBADMIN role:

```sql
=> CREATE ROLE appviewer;
CREATE ROLE
=> GRANT appviewer TO dbadmin;
ROLLBACK 2347: Cannot alter predefined role "dbadmin"
```

You can, however, grant the DBADMIN role to other roles to augment a set of privileges. See [Role Hierarchy](#) for more information.

**View a List of Database Superusers**

To see who is a superuser, run the vsq1 \du command. In this example, only dbadmin is a superuser.

```sql
=> \du
List of users
User name  Is Superuser
-----------------------------------------
dbadmin  t
Fred  f
Bob  f
Sue  f
Alice  f
User1  f
User2  f
User3  f
u1  f
u2  f
(10 rows)
```

**See Also**

[Database Administration User](#)
DBDUSER Role

The DBDUSER role is predefined, and allows non-DBADMIN users to access Database Designer functions. This role must be explicitly granted by a superuser or DBADMIN user.

Note: Non-DBADMIN users with the DBDUSER role cannot run Database Designer through Administration Tools. Only DBADMIN users can run Administration Tools.

You cannot assign any additional privileges to the DBDUSER role, but you can grant the DBDUSER role to other roles to augment a set of privileges.

After you are granted the DBDUSER role, you must enable it before you can run Database Designer using command-line functions. For more information, see About Running Database Designer Programmatically.

Important: When you create a DBADMIN user or grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs. Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Administration Tools or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into some system resource pools for less-intensive tasks.

PSEUDOSUPERUSER Role

The special PSEUDOSUPERUSER role is automatically created in each database. A superuser (or someone with the PSEUDOSUPERUSER role) can perform grant and revoke on this role. The PSEUDOSUPERUSER cannot revoke or change any superuser privileges.

Users with the PSEUDOSUPERUSER role are entitled to complete administrative privileges, including the ability to:

- Create schemas
- Create and grant privileges to roles
- Bypass all GRANT/REVOKE authorization
- Set user account's passwords
- Lock and unlock user accounts
- Create or drop a UDF library
- Create or drop a UDF function
- Create or drop an external procedure
- Add or edit comments on nodes
- Create or drop password profiles

You cannot revoke any of these privileges from a PSEUDOSUPERUSER.

You can assign additional privileges to the PSEUDOSUPERUSER role, but you cannot assign any additional roles; for example, the following is not allowed:

```sql
=> CREATE ROLE appviewer;
CREATE ROLE
=> GRANT appviewer TO pseudosuperuser;
ROLLBACK 2347: Cannot alter predefined role "pseudosuperuser"
```

PUBLIC Role

By default, every database has the special PUBLIC role. Vertica grants this role to each user automatically, and it is automatically enabled. You grant privileges to this role that every user should have by default. You can also grant access to roles to PUBLIC, which allows any user to access the role using the `SET ROLE` statement.

**Note:** The PUBLIC role can never be dropped, nor can it be revoked from users or roles.

Privileges created using the WITH GRANT OPTION cannot be granted to a Public Role:

```sql
=> CREATE TABLE t1(a int);
CREATE TABLE
=> GRANT SELECT on t1 to PUBLIC with grant option;
ROLLBACK 3484: Grant option for a privilege cannot be granted to "public"
```

For more information see [How to Grant Privileges](#).

Example

In the following example, if the superuser hadn't granted INSERT privileges on the table publicdata to the PUBLIC group, the INSERT statement executed by user bob would fail:
CREATE TABLE publicdata (a INT, b VARCHAR);
CREATE TABLE
=> GRANT INSERT, SELECT ON publicdata TO PUBLIC;
GRANT PRIVILEGE
=> CREATE PROJECTION publicdataprop AS (SELECT * FROM publicdata);
CREATE PROJECTION

dbadmin=> \c - bob
You are now connected as user "bob".

=> INSERT INTO publicdata VALUES (10, 'Hello World');
OUTPUT
--------
 1
(1 row)

See Also
PUBLIC User

SYSMONITOR Role

An organization's database administrator may have many responsibilities outside of maintaining Vertica as a DBADMIN user. In this case, as the DBADMIN you may want to delegate some Vertica administrative tasks to another Vertica user.

The DBADMIN can assign a delegate the SYSMONITOR role to grant access to system tables without granting full DBADMIN access.

The SYSMONITOR role provides the following privileges.

- View all system tables that are marked as monitorable. You can see a list of all the monitorable tables by issuing the statement:

  => select * from system_tables where is_monitorable='t';

- If WITH ADMIN OPTION was included when granting SYSMONITOR to the user or role, that user or role can then grant SYSMONITOR privileges to other users and roles.

Grant a SYSMONITOR Role

To grant a user or role the SYSMONITOR role, you must be one of the following:
- a DBADMIN user
- a user assigned the SYSMONITOR who has the ADMIN OPTION

Use the **GRANT (Role)** SQL statement to assign a user the SYSMONITOR role. This example shows how to grant the SYSMONITOR role to user1 and includes administration privileges by using the WITH ADMIN OPTION parameter. The ADMIN OPTION grants the SYSMONITOR role administrative privileges.

```sql
=> GRANT SYSMONITOR TO user1 WITH ADMIN OPTION;
```

This example shows how to revoke the ADMIN OPTION from the SYSMONITOR role for user1:

```sql
=> REVOKE ADMIN OPTION for SYSMONITOR FROM user1;
```

Use **CASCADE** to revoke ADMIN OPTION privileges for all users assigned the SYSMONITOR role:

```sql
=> REVOKE ADMIN OPTION for SYSMONITOR FROM PUBLIC CASCADE;
```

Example

This example shows how to:

- Create a user
- Create a role
- Grant SYSMONITOR privileges to the new role
- Grant the role to the user

```sql
=> CREATE USER user1;
=> CREATE ROLE monitor;
=> GRANT SYSMONITOR to monitor;
=> GRANT monitor to user1;
```

Assign SYSMONITOR Privileges

This example uses the user and role created in the Grant SYSMONITOR Role example and shows how to:
- Create a table called personal_data

- Log in as user1

- Grant user1 the monitor role. (You already granted the monitor SYSMONITOR privileges in the Grant a SYSMONITOR Role example.)

- Run a SELECT statement as user1

The results of the operations are based on the privilege already granted to user1.

```sql
=> CREATE TABLE personal_data (SSN varchar (256));
=> \c -user1;
user1=> SET ROLE monitor;
user1=> SELECT COUNT(*) FROM TABLES;
    COUNT
   -------
     1
(1 row)
```

Because you assigned the SYSMONITOR role, user1 can see the number of rows in the Tables system table. In this simple example, there is only one table (personal_data) in the database so the SELECT COUNT returns one row. In actual conditions, the SYSMONITOR role would see all the tables in the database.

Check if a Table is Accessible by SYSMONITOR

Use the following command to check if a system table can be accessed by a user assigned the SYSMONITOR role:

```sql
=> select table_name, is_monitorable from system_tables where table_name='<table_name>';  
```

Example

This example checks whether the current_session system table is accessible by the SYSMONITOR:

```sql
=> select table_name, is_monitorable from system_tables where table_name='current_session';
table_name | is_monitorable
-------------
current_session | t
```
The t in the is_monitorable column indicates the current_session system table is accessible by the SYSMONITOR.

Default Roles for Database Users

By default, no roles (other than the default PUBLIC Role) are enabled at the start of a user session.

<table>
<thead>
<tr>
<th>SHOW ENABLED_ROLES;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>enabled roles</td>
</tr>
</tbody>
</table>

(1 row)

A superuser can set one or more default roles for a user, which are automatically enabled at the start of the user's session. Setting a default role is a good idea if users normally rely on the privileges granted by one or more roles to carry out the majority of their tasks. To set a default role, use the DEFAULT ROLE parameter of the ALTER USER statement as superuser:

<table>
<thead>
<tr>
<th>ALTER USER Bob DEFAULT ROLE logadmin;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER USER</td>
</tr>
</tbody>
</table>

You are now connected as user "Bob"

<table>
<thead>
<tr>
<th>SHOW ENABLED_ROLES;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>enabled roles</td>
</tr>
</tbody>
</table>

(1 row)

Notes

- Only roles that the user already has access to can be made default.
- Unlike granting a role, setting a default role or roles overwrites any previously-set defaults.
- To clear any default roles for a user, use the keyword NONE as the role name in the DEFAULT ROLE argument.
- Default roles only take effect at the start of a user session. They do not affect the roles enabled in the user's current session.
- Avoid giving users default roles that have administrative or destructive privileges (the PSEUDOSUPERUSER role or DROP privileges, for example). By forcing users to explicitly enable these privileges, you can help prevent accidental data loss.
Using Database Roles

There are several steps to using roles:

1. A superuser creates a role using the `CREATE ROLE` statement.
2. A superuser or object owner grants privileges to the role.
3. A superuser or users with administrator access to the role grant users and other roles access to the role.
4. Users granted access to the role run the `SET ROLE` command to make that role active and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.

Tip: Query system tables `ROLES`, `GRANTS`, and `USERS` to see any directly-assigned roles. Because these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly), see the `HAS_ROLE()` function for additional information.

See Also

- About MC Privileges and Roles

Role Hierarchy

In addition to granting roles to users, you can also grant roles to other roles. This lets you build hierarchies of roles, with more privileged roles (an administrator, for example) being assigned all of the privileges of lesser-privileged roles (a user of a particular application), in addition to the privileges you assign to it directly. By organizing your roles this way, any privilege you add to the application role (reading or writing to a new table, for example) is automatically made available to the more-privileged administrator role.
Example

The following example creates two roles, assigns them privileges, then assigns them to a new administrative role.

1. Create new table applog:

   => CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));

2. Create a new role called logreader:

   => CREATE ROLE logreader;

3. Grant the logreader role read-only access on the applog table:

   => GRANT SELECT ON applog TO logreader;

4. Create a new role called logwriter:

   => CREATE ROLE logwriter;

5. Grant the logwriter write access on the applog table:

   => GRANT INSERT ON applog TO logwriter;

6. Create a new role called logadmin, which will rule the other two roles:

   => CREATE ROLE logadmin;

7. Grant the logadmin role privileges to delete data:

   => GRANT DELETE ON applog TO logadmin;

8. Grant the logadmin role privileges to have the same privileges as the logreader and logwriter roles:

   => GRANT logreader, logwriter TO logadmin;

9. Create new user Bob:
CREATE USER Bob;

10. Give Bob logadmin privileges:

GRANT logadmin TO Bob;

The user Bob can now enable the logadmin role, which also includes the logreader and logwriter roles. Note that Bob cannot enable either the logreader or logwriter role directly. A user can only enable explicitly-granted roles.

Hierarchical roles also works with administrative access to a role:

GRANT logreader, logwriter TO logadmin WITH ADMIN OPTION;
GRANT ROLE logadmin TO Bob;
\c - bob; -- connect as Bob
You are now connected as user "Bob".
GRANT logadmin; -- Enable logadmin role
SET ROLE logadmin;
GRANT logreader TO Alice;

Note that the user Bob only has administrative access to the logreader and logwriter roles through the logadmin role. He doesn't have administrative access to the logadmin role, since it wasn't granted to him with the optional WITH ADMIN OPTION argument:

GRANT logadmin TO Alice;
WARNING: Some roles were not granted
GRANT ROLE

For Bob to be able to grant the logadmin role, a superuser would have had to explicitly grant him administrative access.

See Also

- About MC Privileges and Roles

Creating Database Roles

A superuser creates a new role using the CREATE ROLE statement. Only a superuser can create or drop roles.

CREATE ROLE administrator;
The newly-created role has no privileges assigned to it, and no users or other roles are initially granted access to it. A superuser must grant privileges and access to the role.

Deleting Database Roles

A superuser can delete a role with the DROP ROLE statement.

Note that if any user or other role has been assigned the role you are trying to delete, the DROP ROLE statement fails with a dependency message.

```sql
=> DROP ROLE administrator;
NOTICE: User Bob depends on Role administrator
ROLLBACK: DROP ROLE failed due to dependencies
DETAIL: Cannot drop Role administrator because other objects depend on it
HINT: Use DROP ROLE ... CASCADE to remove granted roles from the dependent users/roles
```

Supply the optional CASCADE parameter to drop the role and its dependencies.

```sql
=> DROP ROLE administrator CASCADE;
DROP ROLE
```

Granting Privileges to Roles

A superuser or owner of a schema, table, or other database object can assign privileges to a role, just as they would assign privileges to an individual user by using the GRANT statements described in the SQL Reference Manual. See About Database Privileges for information about which privileges can be granted.

Granting a privilege to a role immediately affects active user sessions. When you grant a new privilege, it becomes immediately available to every user with the role active.

Example

The following example creates two roles and assigns them different privileges on a single table called applog.

1. Create a table called applog:

```sql
=> CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));
```

2. Create a new role called logreader:
CREATE ROLE logreader;

3. Assign read-only privileges to the logreader role on table applog:

   => GRANT SELECT ON applog TO logreader;

4. Create a role called logwriter:

   => CREATE ROLE logwriter;

5. Assign write privileges to the logwriter role on table applog:

   => GRANT INSERT ON applog TO logwriter;

See the SQL Reference Manual for the different GRANT statements.

Revoking Privileges From Roles

Use one of the REVOKE statements to revoke a privilege from a role.

   => REVOKE INSERT ON applog FROM logwriter;
   REVOKE PRIVILEGE

Revoking a privilege immediately affects any user sessions that have the role active. When you revoke a privilege, it is immediately removed from users that rely on the role for the privilege.

See the SQL Reference Manual for the different REVOKE statements.

Granting Access to Database Roles

A pseudosuperuser or dbadmin user can assign any role to a user or to another role using the GRANT command. The simplest form of this command is:

   GRANT role [, ...] TO { user | role } [, ...]

Vertica returns a NOTICE if you grant a role to a user who has already been granted that role. For example:

   => GRANT commenter to Bob;
   NOTICE 4622: Role "commenter" was already granted to user "Bob"

See GRANT (Role) in the SQL Reference Manual for details.
Example

The following example shows how to create a role called `commenter` and grant that role to user Bob:

1. Create a table called comments.

   ```sql
   => CREATE TABLE comments (id INT, comment VARCHAR);
   ```

2. Create a role called `commenter`.

   ```sql
   => CREATE ROLE commenter;
   ```

3. Grant privileges to the `commenter` role on the comments table.

   ```sql
   => GRANT INSERT, SELECT ON comments TO commenter;
   ```

4. Grant the `commenter` role to user Bob.

   ```sql
   => GRANT commenter TO Bob;
   ```

Before being able to access the role and its associated privileges, Bob must enable the newly-granted role to himself.

1. Connect to the database as user Bob.

   ```sql
   => \c - Bob
   ```

2. Enable the role.

   ```sql
   => SET ROLE commenter;
   ```

3. Insert some values into the comments table.

   ```sql
   => INSERT INTO comments VALUES (1, 'Hello World');
   ```

   Based on the privileges granted to Bob by the `commenter` role, Bob can insert and query the comments table.

4. Query the comments table.
=> SELECT * FROM comments;
<table>
<thead>
<tr>
<th>id</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hello World</td>
</tr>
</tbody>
</table>
(1 row)

5. Commit the transaction.

=> COMMIT;

Note that Bob does not have proper permissions to drop the table.

=> DROP TABLE comments; ROLLBACK
4000: Must be owner of relation comments

See Also

- Granting Database Access to MC Users

Revoking Access From Database Roles

A superuser can revoke any role from a user or from another role using the REVOKE command. The simplest form of this command is:

REVOKE role [, ...] FROM { user | role | PUBLIC } [, ...]

See REVOKE (Role) in the SQL Reference Manual for details.

Example

To revoke access from a role, use the REVOKE (Role) statement:

1. Connect to the database as a superuser:

    \c - dbadmin

2. Revoke the commenter role from user Bob:

    => REVOKE commenter FROM bob;
Granting Administrative Access to a Role

A superuser can assign a user or role administrative access to a role by supplying the optional WITH ADMIN OPTION argument to the GRANT statement. Administrative access allows the user to grant and revoke access to the role for other users (including granting them administrative access). Giving users the ability to grant roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates granting the user bob administrative access to the commenter role, then connecting as bob and granting a role to another user.

1. Connect to the database as a superuser (or a user with administrative access):

   ```
   => \c - dbadmin
   ```

2. Grant administrative options on the commenter role to Bob

   ```
   => GRANT commenter TO Bob WITH ADMIN OPTION;
   ```

3. Connect to the database as user Bob

   ```
   => \c - Bob
   ```

4. As user Bob, grant the commenter role to Alice:

   ```
   => GRANT commenter TO Alice;
   ```

Users with administrative access to a role can also grant other users administrative access:

```
=> GRANT commenter TO alice WITH ADMIN OPTION;
```

As with all user privilege models, database superusers should be cautious when granting any user a role with administrative privileges. For example, if the database superuser grants two users a role with administrative privileges, both users can revoke the role of the other user. This example shows granting the appadmin role (with administrative privileges) to users bob and alice. After each user has been granted the appadmin role, either use can connect as the other will full privileges.
Revoking Administrative Access From a Role

A superuser can revoke administrative access from a role using the ADMIN OPTION parameter with the REVOKE statement. Giving users the ability to revoke roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates revoking administrative access from Alice for the commenter role.

1. Connect to the database as a superuser (or a user with administrative access)

   => \c - dbadmin

2. Issue the REVOKE command with ADMIN OPTION parameters:

   => REVOKE ADMIN OPTION FOR commenter FROM alice;

Enabling Roles

By default, roles aren’t enabled automatically for a user account. (See Default Roles for Database Users for a way to make roles enabled automatically.) Users must explicitly enable a role using the SET ROLE statement. When users enable a role in their session, they gain all of the privileges assigned to that role. Enabling a role does not affect any other roles that the users have active in their sessions. They can have multiple roles enabled simultaneously, gaining the combined privileges of all the roles they have enabled, plus any of the privileges that have been granted to them directly.

   => SELECT * FROM applog;
   ERROR:  permission denied for relation applog

   => SET ROLE logreader;
You can enable all of the roles available to your user account using the SET ROLE ALL statement.

```sql
=> SET ROLE ALL;SET
=> SHOW ENABLED_ROLES;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled roles</td>
<td>logreader, logwriter</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- Viewing a User's Role

Disabling Roles

To disable all roles, use the SET ROLE NONE statement:

```sql
=> SET ROLE NONE;
=> SHOW ENABLED_ROLES;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled roles</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

Viewing Enabled and Available Roles

You can list the roles you have enabled in your session using the SHOW ENABLED ROLES statement:

```sql
=> SHOW ENABLED_ROLES;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
You can find the roles available to your account using the SHOW AVAILABLE ROLES statement:

```sql
Bob=> SHOW AVAILABLE_ROLES;
name | setting
-----------------------------
available roles | logreader, logwriter
(1 row)
```

### Viewing Named Roles

To view the names of all roles users can access, along with any roles that have been assigned to those roles, query the `V_CATALOG.ROLES` system table.

```sql
=> SELECT * FROM roles;
role_id | name         | assigned_roles
---------|--------------|------------------
45035996273704964 | public       |                  
45035996273704966 | dbduser      |                  
45035996273704968 | dbadmin      | dbduser*         
45035996273704972 | pseudosuperuser | dbadmin*       
45035996273704974 | logreader    |                  
45035996273704976 | logwriter    |                  
45035996273704978 | logadmin     | logreader, logwriter
(7 rows)
```

**Note:** An asterisk (*) in the output means that role was granted WITH ADMIN OPTION.

### Viewing a User's Role

The `HAS_ROLE()` function lets you see if a role has been granted to a user.

Non-superusers can check their own role membership using `HAS_ROLE('role_name')`, but only a superuser can look up other users' memberships using the `user_name` parameter. Omitting the `user_name` parameter will return role results for the superuser who is calling the function.

### How to View a User's Role

In this example, user Bob wants to see if he's been assigned the logwriter command. The output returns Boolean value `t` for true, denoting that Bob is assigned the specified logwriter role:
In this example, a superuser wants to verify that the logadmin role has been granted to user Ted:

```
dbadmin=> SELECT HAS_ROLE('Ted', 'logadmin');

HAS_ROLE
--------
t
(1 row)
```

The output returns boolean value `t` for true, denoting that Ted is assigned the specified logadmin role:

```
HAS_ROLE
--------
t
(1 row)
```

Note that if a superuser omits the user_name argument, the function looks up that superuser's role. The following output indicates that this superuser is not assigned the logadmin role:

```
dbadmin=> SELECT HAS_ROLE('logadmin');

HAS_ROLE
--------
f
(1 row)
```

Output of the function call with user Alice indicates that she is not granted the logadmin role:

```
dbadmin=> SELECT HAS_ROLE('Alice', 'logadmin');

HAS_ROLE
--------
f
(1 row)
```

To view additional information about users, roles and grants, you can also query the following system tables in the V_CATALOG schema to show directly-assigned roles:

- **ROLES**
- **GRANTS**
- **USERS**

Note that the system tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). You need to call the HAS_ROLE() function for that information.
**Users**

This command returns all columns from the USERS system table:

```
=> SELECT * FROM users;
-[ RECORD 1 ]
-----------------------------------------
user_id | 45035996273704962
user_name | dbadmin
is_super_user | t
profile_name | default
is_locked | f
lock_time | |
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
all_roles | dbadmin*, pseudosuperuser*default_roles
dbadmin*, pseudosuperuser*
```

**Note:** An asterisk (*) in table output for all_roles and default_roles columns indicates a role granted WITH ADMIN OPTION.

**Roles**

The following command returns all columns from the ROLES system table:

```
=> SELECT * FROM roles;

<table>
<thead>
<tr>
<th>role_id</th>
<th>name</th>
<th>assigned_roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273704964</td>
<td>public</td>
<td></td>
</tr>
<tr>
<td>45035996273704964</td>
<td>dbduser</td>
<td></td>
</tr>
<tr>
<td>45035996273704964</td>
<td>dbadmin</td>
<td>dbduser*</td>
</tr>
<tr>
<td>45035996273704964</td>
<td>pseudosuperuser</td>
<td>dbadmin*</td>
</tr>
</tbody>
</table>
```

**Grants**

The following command returns all columns from the GRANTS system table:

```
=> SELECT * FROM grants;

<table>
<thead>
<tr>
<th>grantor</th>
<th>privileges_description</th>
<th>object_schema</th>
<th>object_name</th>
<th>grantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td>public</td>
<td>public</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td>v_internal</td>
<td>public</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td>v_catalog</td>
<td>public</td>
</tr>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td>v_monitor</td>
<td>public</td>
</tr>
</tbody>
</table>
```

(4 rows)
Viewing User Roles on Management Console

You can see an MC user's roles and database resources through the MC Settings > User management page on the Management Console interface. For more information, see About MC Privileges and Roles.
Using the Administration Tools

The Vertica Administration tools allow you to easily perform administrative tasks. You can perform most Vertica database administration tasks with Administration Tools.

Run Administration Tools using the Database Administrator account on the Administration host, if possible. Make sure that no other Administration Tools processes are running.

If the Administration host is unresponsive, run Administration Tools on a different node in the cluster. That node permanently takes over the role of Administration host.

Any user can view the man page available for admintools. Enter the following:

```
man admintools
```  

Running Administration Tools

As dbadmin user, you can run administration tools. The syntax follows:

```
/opt/vertica/bin/admintools [--debug ][
   { -h | --help }|
   { -a | --help_all}|
   { -t | --tool } name_of_tool[ options]
]
```

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>If you include the debug option, Vertica logs debug information.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You can specify the debug option with or without naming a specific tool. If you specify debug with a specific tool, Vertica logs debug information during tool execution. If you do not specify a tool, Vertica logs debug information when you run tools through the admintools user interface.</td>
</tr>
<tr>
<td>-h --help</td>
<td>Outputs abbreviated help.</td>
</tr>
</tbody>
</table>
| -a --help_all   | Outputs verbose help, which lists all command-line sub-commands and options.

{ -t | --tool } name_of_tool

Specifies the tool to run, where name_of_tool is one of the tools described in the help output, and options are one or more
An unqualified `admintools` command displays the Main Menu dialog box.

If you are unfamiliar with this type of interface, read Using the Administration Tools Interface

**First Login as Database Administrator**

The first time you log in as the Database Administrator and run the Administration Tools, the user interface displays.

1. In the end-user license agreement (EULA) window, type `accept` to proceed.
   
   A window displays, requesting the location of the license key file you downloaded from the Vertica Web site. The default path is `/tmp/vlicense.dat`.

2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click OK.

**Between Dialogs**

While the Administration Tools are working, you see the command line processing in a window similar to the one shown below. Do not interrupt the processing.
**Using the Administration Tools Interface**

The Vertica Administration Tools are implemented using Dialog, a graphical user interface that works in terminal (character-cell) windows. The interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. Thus, this section describes how to use the Administration Tools using only keystrokes.

**Note:** This section does not describe every possible combination of keystrokes you can use to accomplish a particular task. Feel free to experiment and to use whatever keystrokes you prefer.

**Enter [Return]**

In all dialogs, when you are ready to run a command, select a file, or cancel the dialog, press the Enter key. The command descriptions in this section do not explicitly instruct you to press Enter.
OK - Cancel - Help

The OK, Cancel, and Help buttons are present on virtually all dialogs. Use the tab, space bar, or right and left arrow keys to select an option and then press Enter. The same keystrokes apply to dialogs that present a choice of Yes or No.

Menu Dialogs

Some dialogs require that you choose one command from a menu. Type the alphanumeric character shown or use the up and down arrow keys to select a command and then press Enter.

List Dialogs

In a list dialog, use the up and down arrow keys to highlight items, then use the space bar to select the items (which marks them with an X). Some list dialogs allow you to select multiple items. When you have finished selecting items, press Enter.
Form Dialogs

In a form dialog (also referred to as a dialog box), use the tab key to cycle between OK, Cancel, Help, and the form field area. Once the cursor is in the form field area, use the up and down arrow keys to select an individual field (highlighted) and enter information. When you have finished entering information in all fields, press Enter.

Help Buttons

Online help is provided in the form of text dialogs. If you have trouble viewing the help, see Notes for Remote Terminal Users.

K-Safety Support in Administration Tools

The Administration Tools allow certain operations on a K-Safe database, even if some nodes are unresponsive.

The database must have been marked as K-Safe using the MARK_DESIGN_KSAFE function.

The following management functions within the Administration Tools are operational when some nodes are unresponsive.

Note: Vertica users can perform much of the below functionality using the Management Console interface. See Management Console and Administration Tools for details.
- View database cluster state
- Connect to database
- Start database (including manual recovery)
- Stop database
- Replace node (assuming node that is down is the one being replaced)
- View database parameters
- Upgrade license key

The following operations work with unresponsive nodes; however, you might have to repeat the operation on the failed nodes after they are back in operation:

- Distribute config files
- Install external procedure
- (Setting) database parameters

The following management functions within the Administration Tools require that all nodes be UP in order to be operational:

- Create database
- Run the Database Designer
- Drop database
- Set restart policy
- Roll back database to Last Good Epoch

**Notes for Remote Terminal Users**

The appearance of the graphical interface depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on a Windows platform.
Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, make sure your window is at least 81 characters wide and 23 characters high.

If you are using PuTTY, you can make the Administration Tools look like the screen captures in this document:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Appearance.
4. In the Font settings, click the Change... button.
5. Select Font: Courier New: Regular Size: 10
6. Click Apply.

Repeat these steps for each existing session that you use to run the Administration Tools.

You can also change the translation to support UTF-8:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Translation.
4. In the "Received data assumed to be in which character set" drop-down menu, select UTF-8.
5. Click Apply.

**Using Administration Tools Help**

The Help on Using the Administration Tools command displays a help screen about using the Administration Tools.
Most of the online help in the Administration Tools is context-sensitive. For example, if you use up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

In a Menu Dialog

1. Use the up and down arrow keys to choose the command for which you want help.
2. Use the Tab key to move the cursor to the Help button.

3. Press Enter (Return).

In a Dialog Box

1. Use the up and down arrow keys to choose the field on which you want help.

2. Use the Tab key to move the cursor to the Help button.

3. Press Enter (Return).

Scrolling

Some help files are too long for a single screen. Use the up and down arrow keys to scroll through the text.

Password Authentication

When you create a new user with the CREATE USER command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the ALTER USER command.
See Security and Authentication for more information about controlling database authorization through passwords.

Tip: Unless the database is used solely for evaluation purposes, Vertica recommends that all database users have encrypted passwords.

Distributing Changes Made to the Administration Tools Metadata

Administration Tools-specific metadata for a failed node will fall out of synchronization with other cluster nodes if you make the following changes:

- Modify the restart policy
- Add one or more nodes
- Drop one or more nodes.

When you restore the node to the database cluster, you can use the Administration Tools to update the node with the latest Administration Tools metadata:

1. Log on to a host that contains the metadata you want to transfer and start the Administration Tools. (See Using the Administration Tools.)

2. On the Main Menu in the Administration Tools, select Configuration Menu and click OK.

3. On the Configuration Menu, select Distribute Config Files and click OK.


   The Administration Tools metadata is distributed to every host in the cluster.

5. Restart the database.

Administration Tools and Management Console

You can perform most database administration tasks using the Administration Tools, but you have the additional option of using the more visual and dynamic Management Console.
The following table compares the functionality available in both interfaces. Continue to use Administration Tools and the command line to perform actions not yet supported by Management Console.

<table>
<thead>
<tr>
<th>Vertica Functionality</th>
<th>Management Console</th>
<th>Administration Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a Web interface for the administration of Vertica</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Manage/monitor one or more databases and clusters through a UI</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Manage multiple databases on different clusters</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View database cluster state</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View multiple cluster states</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Connect to the database</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Start/stop an existing database</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stop/restart Vertica on host</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Kill a Vertica process on host</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Create one or more databases</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View databases</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Remove a database from view</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Drop a database</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Create a physical schema design (Database Designer)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modify a physical schema design (Database Designer)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Set the restart policy</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Roll back database to the Last Good Epoch</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Manage clusters (add, replace, remove hosts)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Rebalance data across nodes in the database</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Vertica Functionality</td>
<td>Management Console</td>
<td>Administration Tools</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Configure database parameters dynamically</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>View database activity in relation to physical resource usage</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>View alerts and messages dynamically</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>View current database size usage statistics</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>View database size usage statistics over time</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Upload/upgrade a license file</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Warn users about license violation on login</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Create, edit, manage, and delete users/user information</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Use LDAP to authenticate users with company credentials</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Manage user access to MC through roles</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Map Management Console users to a Vertica database</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Enable and disable user access to MC and/or the database</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Audit user activity on database</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Hide features unavailable to a user through roles</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Generate new user (non-LDAP) passwords</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Management Console Provides some, but Not All of the Functionality Provided By the Administration Tools. MC Also Provides Functionality Not Available in the Administration Tools.

See Also

- Monitoring Vertica Using Management Console
Administration Tools Reference

With Vertica Administration Tools, you can perform the following tasks:

- View the database cluster state
- Connect to the database
- Start the database
- Stop the database
- Write scripts

Viewing Database Cluster State

This tool shows the current state of the nodes in the database.

1. On the Main Menu, select View Database Cluster State, and click OK.
   The normal state of a running database is ALL UP. The normal state of a stopped database is ALL DOWN.
2. If some hosts are UP and some DOWN, restart the specific host that is down using Restart Vertica on Host from the Administration Tools, or you can start the database as described in Starting and Stopping the Database (unless you have a known node failure and want to continue in that state.)

Nodes shown as INITIALIZING or RECOVERING indicate that Failure Recovery is in progress.
Nodes in other states (such as NEEDS_CATCHUP) are transitional and can be ignored unless they persist.

See Also

- Advanced Menu Options

Connecting to the Database

This tool connects to a running database with vsql. You can use the Administration Tools to connect to a database from any node within the database while logged in to any user account with access privileges. You cannot use the Administration Tools to connect from a host that is not a database node. To connect from other hosts, run vsql as described in Connecting from the Command Line.

1. On the Main Menu, click Connect to Database, and then click OK.

2. Supply the database password if asked:

   ```
   Password:
   ```

When you create a new user with the CREATE USER command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the ALTER USER command.

The Administration Tools connect to the database and transfer control to vsql.

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.
Type: \h or \? for help with vsql commands
     \g or terminate with semicolon to execute query
     \q to quit
=>
```

See Using vsql for more information.

Note: After entering your password, you may be prompted to change your password if it has expired. See Implementing Client Authentication for details of password security.
See Also

- **CREATE USER**
- **ALTER USER**

### Restarting Vertica on Host

This tool restarts the Vertica process on one or more nodes in a running database. Use this tool when a cluster host reboots while the database is running. The spread daemon starts automatically but the Vertica process does not, so the node does not automatically rejoin the cluster.

1. On the Main Menu, select View Database Cluster State, and click OK.
2. If one or more nodes are down, select Restart Vertica on Host, and click OK.
3. Select the database that contains the host that you want to restart, and click OK.
4. Select the Host to restart, and click OK.
5. Select View Database Cluster State again to verify all nodes are up.

![Screen capture showing DB, Host, and State columns with Stock_Schema as an example]
Configuration Menu Item

The Configuration Menu includes:

Creating a Database

Use the procedures below to create either an Enterprise Mode or Eon Mode Beta database.

Create an Enterprise Mode Database

1. On the Configuration Menu, click Create Database and then click OK.
2. Select Enterprise Mode as your database mode.
3. Enter the name of the database and an optional comment. Click OK.
4. Enter a password. See Creating a Database Name and Password for rules.

    If you do not enter a password, you are prompted to indicate whether you want to enter a password. Click Yes to enter a password or No to create a database without a superuser password.

    Caution: If you do not enter a password at this point, superuser password is set to empty. Unless the database is for evaluation or academic purposes, Vertica strongly recommends that you enter a superuser password.

5. If you entered a password, enter the password again.
6. Select the hosts to include in the database. The hosts in this list are the ones that were specified at installation time (install_vertica -s).
7. Specify the directories in which to store the catalog and data files.

    Database data directories
    Catalog path name: /home/dbadmin
    Data path name: /home/dbadmin

    Note: Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

    Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

8. Check the current database definition for correctness, and click Yes to proceed.
9. A message indicates that you have successfully created a database. Click OK.

Create an Eon Mode Beta Database

1. On the Configuration Menu, click Create Database and then click OK.

2. Select Eon Mode as your database mode.

3. Enter the name of the database and an optional comment. Click OK.

4. Enter a password. See Creating a Database Name and Password for rules.

   If you do not enter a password, you are prompted to indicate whether you want to enter a password. Click Yes to enter a password or No to create a database without a superuser password.

   **Caution:** If you do not enter a password at this point, superuser password is set to empty. Unless the database is for evaluation or academic purposes, Vertica strongly recommends that you enter a superuser password.

5. If you entered a password, enter the password again.
6. Select the hosts to include in the database. The hosts in this list are the ones that were specified at installation time (install_vertica -s).

7. Specify the directories in which to store the catalog and depot, depot size, communal storage location, and number of shards.
   - Depot Size: Use an integer followed by %, K, G, or T. Default is 80% of total disk size.
   - Communal Storage: Use an existing Amazon S3 bucket in the same region as your instances. Specify a new subfolder name, which Vertica will dynamically create within the existing S3 bucket. For example, s3://existingbucket/newstorage1. You can create a new subfolder within existing ones, but database creation will roll back if you do not specify any new subfolder name.
   - Number of Shards: Use a whole number. Default is equal to the number of nodes.

![Database Data Directories](image)

Note: Catalog and depot paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

8. Check the current database definition for correctness, and click Yes to proceed.

9. A message indicates that you have successfully created a database. Click OK.

Dropping a Database

This tool drops an existing database. Only the Database Administrator is allowed to drop a database.

1. Stop the database.

2. On the Configuration Menu, click Drop Database and then click OK.

3. Select the database to drop and click OK.
4. Click Yes to confirm that you want to drop the database.

5. Type yes and click OK to reconfirm that you really want to drop the database.

6. A message indicates that you have successfully dropped the database. Click OK.

When Vertica drops the database, it also automatically drops the node definitions that refer to the database. The following exceptions apply:

- Another database uses a node definition. If another database refers to any of these node definitions, none of the node definitions are dropped.

- A node definition is the only node defined for the host. (Vertica uses node definitions to locate hosts that are available for database creation, so removing the only node defined for a host would make the host unavailable for new databases.)

**Viewing a Database**

This tool displays the characteristics of an existing database.

1. On the Configuration Menu, select View Database and click OK.

2. Select the database to view.

3. Vertica displays the following information about the database:
   - The name of the database.
   - The name and location of the log file for the database.
   - The hosts within the database cluster.
   - The value of the restart policy setting.
Note: This setting determines whether nodes within a K-Safe database are restarted when they are rebooted. See Setting the Restart Policy.

- The database port.
- The name and location of the catalog directory.

**Setting the Restart Policy**

The Restart Policy enables you to determine whether or not nodes in a K-Safe database are automatically restarted when they are rebooted. Since this feature does not automatically restart nodes if the entire database is DOWN, it is not useful for databases that are not K-Safe.

To set the Restart Policy for a database:

1. Open the Administration Tools.
2. On the Main Menu, select Configuration Menu, and click OK.
3. In the Configuration Menu, select Set Restart Policy, and click OK.
4. Select the database for which you want to set the Restart Policy, and click OK.
5. Select one of the following policies for the database:
   - **Never** — Nodes are never restarted automatically.
   - **K-Safe** — Nodes are automatically restarted if the database cluster is still UP. This is the default setting.
   - **Always** — Node on a single node database is restarted automatically.

   *Note: Always does not work if a single node database was not shutdown cleanly or crashed.*

6. Click OK.

**Best Practice for Restoring Failed Hardware**

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:
1. Reboot the machine into runlevel 1, which is a root and console-only mode.

   Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.

2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

   Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

   At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, Vertica will delete all files associated with that projection, which could result in data loss.

**Installing External Procedure Executable Files**

1. Run the Administration Tools.

   $ /opt/vertica/bin/adminTools

2. On the AdminTools Main Menu, click Configuration Menu, and then click OK.

3. On the Configuration Menu, click Install External Procedure and then click OK.

4. Select the database on which you want to install the external procedure.

5. Either select the file to install or manually type the complete file path, and then click OK.

6. If you are not the superuser, you are prompted to enter your password and click OK.

   The Administration Tools automatically create the `<database_catalog_path>/procedures` directory on each node in the database and installs the external procedure in these directories for you.

7. Click OK in the dialog that indicates that the installation was successful.
Advanced Menu Options

This Advanced Menu includes:

Rolling Back the Database to the Last Good Epoch

Vertica provides the ability to roll the entire database back to a specific epoch primarily to assist in the correction of human errors during data loads or other accidental corruptions. For example, suppose that you have been performing a bulk load and the cluster went down during a particular `COPY` command. You might want to discard all epochs back to the point at which the previous `COPY` command committed and run the one that did not finish again. You can determine that point by examining the log files (see Monitoring the Log Files).

1. On the Advanced Menu, select Roll Back Database to Last Good Epoch.
2. Select the database to roll back. The database must be stopped.
3. Accept the suggested restart epoch or specify a different one.
4. Confirm that you want to discard the changes after the specified epoch.

The database restarts successfully.

**Important:** The default value of `HistoryRetentionTime` is 0, which means that Vertica only keeps historical data when nodes are down. This settings prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch. Vertica cannot roll back to an epoch that precedes the AHM.

If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data. For example:

```
=> ALTER DATABASE mydb SET HistoryRetentionTime = 86400;
```

Stopping Vertica on Host

This command attempts to gracefully shut down the Vertica process on a single node.

**Caution:** Do not use this command to shut down the entire cluster. Instead, stop the database to perform a clean shutdown that minimizes data loss.
1. On the Advanced Menu, select Stop Vertica on Host and click OK.

2. Select the hosts to stop.

3. Confirm that you want to stop the hosts.

If the command succeeds View Database Cluster State shows that the selected hosts are DOWN.
If the command fails to stop any selected nodes, proceed to Killing Vertica Process on Host.

### Killing the Vertica Process on Host

This command sends a kill signal to the Vertica process on a node.

**Caution:** Use this command only after you tried to stop the database and stop Vertica on a node and both were unsuccessful.

1. On the Advanced menu, select Kill Vertica Process on Host and click OK.
2. Select the hosts on which to kills the Vertica process.
3. Confirm that you want to stop the processes.

4. If the command succeeds, View Database Cluster State shows that the selected hosts are DOWN.
Upgrading a Vertica License Key

The following steps are for licensed Vertica users. Completing the steps copies a license key file into the database. See Managing Licenses for more information.

1. On the Advanced menu select Upgrade License Key. Click OK.

2. Select the database for which to upgrade the license key.

3. Enter the absolute pathname of your downloaded license key file (for example, /tmp/vlicense.dat). Click OK.

4. Click OK when you see a message indicating that the upgrade succeeded.

Note: If you are using Vertica Community Edition, follow the instructions in Vertica License Changes for instructions to upgrade to a Vertica Premium Edition license key.

Managing Clusters

Cluster Management lets you add, replace, or remove hosts from a database cluster. These processes are usually part of a larger process of adding, removing, or replacing a database node.

Note: View the database state to verify that it is running. See View Database Cluster State. If the database isn't running, restart it. See Starting the Database.
Using Cluster Management

To use Cluster Management:

1. From the Main Menu, select Advanced Menu, and then click OK.
2. In the Advanced Menu, select Cluster Management, and then click OK.
3. Select one of the following, and then click OK.
   - Add Hosts to Database: See Adding Hosts to a Database.
   - Re-balance Data: See Rebalancing Data.
   - Replace Host: See Replacing Hosts.
   - Remove Host from Database: See Removing Hosts from a Database.

Using Administration Tools

The Help Using the Administration Tools command displays a help screen about using the Administration Tools.

Most of the online help in the Administration Tools is context-sensitive. For example, if you use up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

Administration Tools Metadata

The Administration Tools configuration data (metadata) contains information that databases need to start, such as the hostname/IP address of each participating host in the database cluster.

To facilitate hostname resolution within the Administration Tools, at the command line, and inside the installation utility, Vertica enforces all hostnames you provide through the Administration Tools to use IP addresses:

- During installation

  Vertica immediately converts any hostname you provide through command line options --hosts, -add-hosts or --remove-hosts to its IP address equivalent.
If you provide a hostname during installation that resolves to multiple IP addresses (such as in multi-homed systems), the installer prompts you to choose one IP address.

- Vertica retains the name you give for messages and prompts only; internally it stores these hostnames as IP addresses.

- **Within the Administration Tools**

  All hosts are in IP form to allow for direct comparisons (for example db = database = database.example.com).

- **At the command line**

  Vertica converts any hostname value to an IP address that it uses to look up the host in the configuration metadata. If a host has multiple IP addresses that are resolved, Vertica tests each IP address to see if it resides in the metadata, choosing the first match. No match indicates that the host is not part of the database cluster.

Metadata is more portable because Vertica does not require the names of the hosts in the cluster to be exactly the same when you install or upgrade your database.

## Writing Administration Tools Scripts

You can invoke most Administration Tools from the command line or a shell script.

### Syntax

```bash
/opt/vertica/bin/admintools { 
   { -h | --help } 
   | { -a | --help_all} 
   | { [--debug ] { -t | --tool } toolname [ tool-args ] }
}
```

**Note:** For convenience, add `/opt/vertica/bin` to your search path.

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Outputs abbreviated help.</td>
</tr>
<tr>
<td>-a</td>
<td>Outputs verbose help, which lists all command-line sub-commands</td>
</tr>
</tbody>
</table>
-help_all and options.

[debug]
{ -t | -tool }
toolname [args]

Specifies the tool to run, where toolname is one of the tools listed in the help output described below, and args is one or more comma-delimited toolname arguments. If you include the debug option, Vertica logs debug information during tool execution.

Tools

To return a list of all available tools, enter admintools -h at a command prompt.

Note: To create a database or password, see Creating a Database Name and Password for naming rules.

To display help for a specific tool and its options or commands, qualify the specified tool name with --help or -h, as shown in the example below:

```
$ admintools -t connect_db --help
Usage: connect_db [options]

Options:
  -h, --help           show this help message and exit
  -d DB, --database=DB  Name of database to connect
  -p DBPASSWORD, --password=DPASSWORD
                        Database password in single quotes
```

To list all available tools and their commands and options in individual help text, enter admintools -a.

```
$ admintools -a
Usage:
  admintools [-t | --tool] toolName [options]
Valid tools are:
  command_host
  connect_db
  create_db
  database_parameters
  db_add_node
  db_remove_node
  db_replace_node
  db_status
  distribute_config_files
  drop_db
  host_to_node
  install_package
  install_procedure
  kill_host
  kill_node
```
license_audit
list_allnodes
list_db
list_host
list_node
list_packages
logrotate
node_map
re_ip
rebalance_data
restart_db
restart_node
return_epoch
revive_db
set_restart_policy
set_ssl_params
show_active_db
start_db
stop_db
stop_host
stop_node
uninstall_package
upgrade_license_key
view_cluster

Usage: command_host [options]

Options:
-h, --help show this help message and exit
-c CMD, --command=CMD Command to run

Usage: connect_db [options]

Options:
-h, --help show this help message and exit
-d DB, --database=DB Name of database to connect
-p DBPASSWORD, --password=DBPASSWORD Database password in single quotes

Usage: create_db [options]

Options:
-h, --help show this help message and exit
-D DATA, --data_path=DATA Path of data directory[optional] if not using compat21
-c CATALOG, --catalog_path=CATALOG Path of catalog directory[optional] if not using compat21
--compat21 (deprecated) Use Vertica 2.1 method using node names instead of hostnames
-d DB, --database=DB Name of database to be created
-l LICENSEFILE, --license=LICENSEFILE Database license [optional]
-p DBPASSWORD, --password=DBPASSWORD Database password in single quotes [optional]
-P POLICY, --policy=POLICY Database restart policy [optional]
-s NODES, --hosts=NODES comma-separated list of hosts to participate in
database
--skip-fs-checks  Skip file system checks while creating a database (not recommended).
--shard-count=SHARD_COUNT  [Eon only] Number of shards in the database
--communal-storage-location=COMMUNAL_STORAGE_LOCATION  [Eon only] Location of communal storage
--depot-path=DEPOT_PATH  [Eon only] Path to depot directory
--depot-size=DEPOT_SIZE  [Eon only] Size of depot
----------------------------------------------------------------------------------------------------------------------------------------
Usage: database_parameters [options]

Options:
-h, --help  show this help message and exit
-d DB, --database=DB  Name of database
-P PARAMETER, --parameter=PARAMETER  Database parameter
-c COMPONENT, --component=COMPONENT  Component[optional]
-s SUBCOMPONENT, --subcomponent=SUBCOMPONENT  Sub Component[optional]
-p PASSWORD, --password=PASSWORD  Database password[optional]
----------------------------------------------------------------------------------------------------------------------------------------
Usage: db_add_node [options]

Options:
-h, --help  show this help message and exit
-d DB, --database=DB  Name of database to be restarted
-s HOSTS, --hosts=HOSTS  Comma separated list of hosts to add to database
-p DBPASSWORD, --password=DBPASSWORD  Database password in single quotes
-a AHOSTS, --add=AHOSTS  Comma separated list of hosts to add to database
--timeout=NONINTERACTIVE_TIMEOUT  set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -i)
-i, --noprompts  do not stop and wait for user input(default false). Setting this implies a timeout of 20 min.
--compat21  (deprecated) Use Vertica 2.1 method using node names instead of hostnames
--skip-fs-checks  Skip file system checks while adding nodes (not recommended).
----------------------------------------------------------------------------------------------------------------------------------------
Usage: db_remove_node [options]

Options:
-h, --help  show this help message and exit
-d DB, --database=DB  Name of database to be modified
-s HOSTS, --hosts=HOSTS  Name of the host to remove from the db
-p DBPASSWORD, --password=DBPASSWORD  Database password in single quotes
--timeout=NONINTERACTIVE_TIMEOUT  set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets
-i) do not stop and wait for user input (default false).
Setting this implies a timeout of 20 min.
--compat21 (deprecated) Use Vertica 2.1 method using node names instead of hostnames

Usage: db_replace_node [options]

Options:
-h, --help show this help message and exit
-d DB, --database=DB Name of database to be restarted
-o ORIGINAL, --original=ORIGINAL Name of host you wish to replace
-n NEWHOST, --new=NEWHOST Name of the replacement host
-p DBPASSWORD, --password=DBPASSWORD Database password in single quotes
--timeout=NONINTERACTIVE_TIMEOUT set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -i)
-i, --nomprompts do not stop and wait for user input (default false). Setting this implies a timeout of 20 min.
--skip-fs-checks Skip file system checks while replacing nodes (not recommended).

Usage: db_status [options]

Options:
-h, --help show this help message and exit
-s STATUS, --status=STATUS Database status UP, DOWN or ALL (list running dbs - UP, list down dbs - DOWN list all dbs - ALL)

Usage: distribute_config_files
Sends admintools.conf from local host to all other hosts in the cluster

Options:
-h, --help show this help message and exit

Usage: drop_db [options]

Options:
-h, --help show this help message and exit
-d DB, --database=DB Database to be dropped

Usage: host_to_node [options]

Options:
-h, --help show this help message and exit
-s HOST, --host=HOST comma separated list of hostnames which is to be converted into its corresponding nodenames
-d DB, --database=DB show only node/host mapping for this database.

Usage: admintools -t install_package --package PACKAGE -d DB -p PASSWORD
Examples:
admintools -t install_package -d mydb -p 'mypassword' --package default
# (above) install all default packages that aren't currently installed
admintools -t install_package -d mydb -p 'mypassword' --package default --force-reinstall
# (above) upgrade (re-install) all default packages to the current version

admintools -t install_package -d mydb -p 'mypasswd' --package hcat
# (above) install package hcat

See also: admintools -t list_packages

Options:
- `-h, --help`	show this help message and exit
- `-d DBNAME, --dbname=DBNAME`
  	database name
- `-p PASSWORD, --password=PASSWORD`
  	database admin password
- `-P PACKAGE, --package=PACKAGE`
  	specify package or 'all' or 'default'
- `--force-reinstall`	Force a package to be re-installed even if it is already installed.

---------------------------------------------------------------------
Usage: install_procedure [options]

Options:
- `-h, --help`	show this help message and exit
- `-d DBNAME, --dbname=DBNAME`
  	Name of database for installed procedure
- `-f PROCPATH, --file=PROCPATH`
  	Path of procedure file to install
- `-p OWNERPASSWORD, --password=OWNERPASSWORD`
  	Password of procedure file owner

---------------------------------------------------------------------
Usage: kill_host [options]

Options:
- `-h, --help`	show this help message and exit
- `-s HOSTS, --hosts=HOSTS`
  	comma-separated list of hosts on which the vertica process is to be killed using a SIGKILL signal
- `--compat21`
  	(deprecated) Use Vertica 2.1 method using node names instead of hostnames

---------------------------------------------------------------------
Usage: kill_node [options]

Options:
- `-h, --help`	show this help message and exit
- `-s HOSTS, --hosts=HOSTS`
  	comma-separated list of hosts on which the vertica process is to be killed using a SIGKILL signal
- `--compat21`
  	(deprecated) Use Vertica 2.1 method using node names instead of hostnames

---------------------------------------------------------------------
Usage: license_audit --dbname DB_NAME [OPTIONS]
Runs audit and collects audit results.

Options:
- `-h, --help`	show this help message and exit
- `-d DATABASE, --database=DATABASE`
  	Name of the database to retrieve audit results
- `-p PASSWORD, --password=PASSWORD`
  	Password for database admin
- `-q, --quiet`	Do not print status messages.
- `-f FILE, --file=FILE`	Output results to FILE.
Usage: list_allnodes [options]

Options:
- h, --help  show this help message and exit

Usage: list_db [options]

Options:
- h, --help  show this help message and exit
- d DB, --database=DB  Name of database to be listed

Usage: list_host [options]

Options:
- h, --help  show this help message and exit

Usage: list_node [options]

Options:
- h, --help  show this help message and exit
- n NODENAME, --node=NODENAME
  Name of the node to be listed

Usage: admintools -t list_packages [OPTIONS]
Examples:
admintools -t list_packages
admintools -t list_packages --package all
admintools -t list_packages --package default
admintools -t list_packages -d mydb --password 'mypassword'

Options:
- h, --help  show this help message and exit
- d DBNAME, --dbname=DBNAME  database name
- p PASSWORD, --password=PASSWORD  database admin password
- P PACKAGE, --package=PACKAGE
  specify package or 'all' or 'default'

Usage: logrotateconfig [options]

Options:
- h, --help  show this help message and exit
- d DBNAME, --dbname=DBNAME  database name
- r ROTATION, --rotation=ROTATION
  set how often the log is rotated.[daily|weekly|monthly ]
- s MAXLOGSZ, --maxsize=MAXLOGSZ
  set maximum log size before rotation is forced.
- k KEEP, --keep=KEEP
  set # of old logs to keep

Usage: node_map [options]

Options:
- h, --help  show this help message and exit
- d DB, --database=DB  List only data for this database.

Usage: re_ip [options]
Replaces the IP addresses of hosts and databases in a cluster, or changes the control messaging mode/addresses of a database.

Options:
- `-h, --help` show this help message and exit
- `-f MAPFILE, --file=MAPFILE` A text file with IP mapping information. If the `-O` option is not used, the command replaces the IP addresses of the hosts in the cluster and all databases for those hosts. In this case, the format of each line in MAPFILE is: [oldIPaddress newIPaddress] or [oldIPaddress newIPaddress, newControlAddress, newControlBroadcast]. If the former, 'newControlAddress' and 'newControlBroadcast' would set to default values. Usage: `$ admintools -t re_ip -f <mapfile>`
- `-O, --db-only` Updates the control messaging addresses of a database. Also used for error recovery (when Re-IP encounters some certain errors, a mapfile is auto-generated). Format of each line in MAPFILE: [nodeName AssociatedNodeIPaddress, newControlAddress, newControlBroadcast]. 'nodeName' and 'AssociatedNodeIPaddress' must be consistent with admintools.conf. Usage: `$ admintools -t re_ip -f <mapfile> -O -d <db_name>`
- `-i, --noprompts` System does not prompt for the validation of the new settings before performing the Re-IP. Prompting is on by default.
- `-T, --point-to-point` Sets the control messaging mode of a database to point-to-point. Usage: `$ admintools -t re_ip -d <db_name> -T`
- `-U, --broadcast` Sets the control messaging mode of a database to broadcast. Usage: `$ admintools -t re_ip -d <db_name> -U`
- `-d DB, --database=DB` Name of a database. Required with the following options: -O, -T, -U.

Usage: rebalance_data [options]

Options:
- `-h, --help` show this help message and exit
- `-d DBNAME, --dbname=DBNAME` database name
- `-k KSafety, --ksafety=KSAFETY` specify the new k value to use
- `-p PASSWD, --password=PASSWD` Don't re-balance the data, just provide a script for later use.

Usage: restart_db [options]

Options:
- `-h, --help` show this help message and exit
- `-d DB, --database=DB` Name of database to be restarted
- `-e EPOCH, --epoch=EPOCH` Epoch at which the database is to be restarted. If 'last' is given as argument the db is restarted from the last good epoch.
- `-p DBPASSWD, --password=DBPASSWD`
Database password in single quotes

-`k`, `--allow-fallback-keygen`
Generate spread encryption key from Vertica. Use under support guidance only.

-`--timeout=NONINTERACTIVE_TIMEOUT`
set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -i)

-`-i`, `--noprarets`
do not stop and wait for user input (default false).
Setting this implies a timeout of 20 min.

Usage: restart_node [options]

Options:
-`-h`, `--help`
  show this help message and exit
-`-s NODES, --hosts=NODES`
  comma-separated list of hosts to be restarted
-`-d DB, --database=DB`
  Name of database whose node is to be restarted
-`-p DBPASSWORD, --password=DBPASSWORD`
  Database password in single quotes

-`--timeout=NONINTERACTIVE_TIMEOUT`
set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -i)

-`-i`, `--noprarets`
do not stop and wait for user input (default false).
Setting this implies a timeout of 20 min.

-`-F`, `--force`
  force the node to start and auto recover if necessary
-`--compat21`
  (deprecated) Use Vertica 2.1 method using node names instead of hostnames

Usage: return_epoch [options]

Options:
-`-h`, `--help`
  show this help message and exit
-`-d DB, --database=DB`
  Name of database

Usage: revive_db [options]

Options:
-`-h`, `--help`
  show this help message and exit
-`-s NODES, --hosts=NODES`
  comma-separated list of hosts to participate in database

-`--communal-storage-location=COMMUNAL_STORAGE_LOCATION`
  Location of communal storage
-`-d DBNAME, --database=DBNAME`
  Name of database to be revived
-`--force`
  Force cleanup of existing catalog directory
-`--display-only`
  Describe the database on communal storage, and exit

Usage: set_restart_policy [options]

Options:
-`-h`, `--help`
  show this help message and exit
-`-d DB, --database=DB`
  Name of database for which to set policy
-`-p POLICY, --policy=POLICY`
  Restart policy: ('never', 'k safe', 'always')

Usage: set_ssl_params [options]
Options:
- -h, --help show this help message and exit
- -d DB, --database=DB Name of database whose parameters will be set
- -k KEYFILE, --ssl-key-file=KEYFILE Path to SSL private key file
- -c CERTFILE, --ssl-certificate-file=CERTFILE Path to SSL certificate file
- -a CAFILE, --ssl-ca-file=CAFILE Path to SSL CA file
- -p DBPASSWORD, --password=DBPASSWORD Database password in single quotes

Usage: show_active_db [options]

Options:
- -h, --help show this help message and exit

Usage: start_db [options]

Options:
- -h, --help show this help message and exit
- -d DB, --database=DB Name of database to be started
- -p DBPASSWORD, --password=DBPASSWORD Database password in single quotes
- --timeout=NONINTERACTIVE_TIMEOUT set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -1)
- -i, --noprompts do not stop and wait for user input (default false). Setting this implies a timeout of 20 min.
- -F, --force force the database to start at an epoch before data consistency problems were detected.
- -U, --unsafe Start database unsafely, skipping recovery. Use under support guidance only.
- -k, --allow-fallback-keygen Generate spread encryption key from Vertica. Use under support guidance only.

Usage: stop_db [options]

Options:
- -h, --help show this help message and exit
- -d DB, --database=DB Name of database to be stopped
- -p DBPASSWORD, --password=DBPASSWORD Database password in single quotes
- -H, --hibernate [Eon only] Hibernate the database (instead of simply shutting down)
- -F, --force Force the databases to shutdown, even if users are connected.
- --timeout=NONINTERACTIVE_TIMEOUT set a timeout (in seconds) to wait for actions to complete ('never') will wait forever (implicitly sets -1)
- -i, --noprompts do not stop and wait for user input (default false). Setting this implies a timeout of 20 min.

Usage: stop_host [options]

Options:
- -h, --help show this help message and exit
-s HOSTS, --hosts=HOSTS  
    comma-separated list of hosts on which the vertica process is to be killed using a SIGTERM signal

--compat21  
    (deprecated) Use Vertica 2.1 method using node names instead of hostnames

Usage: stop_node [options]

Options:
- h, --help  
  show this help message and exit
- s HOSTS, --hosts=HOSTS  
  comma-separated list of hosts on which the vertica process is to be killed using a SIGTERM signal
--compat21  
    (deprecated) Use Vertica 2.1 method using node names instead of hostnames

Usage: uninstall_package [options]

Options:
- h, --help  
  show this help message and exit
- d DBNAME, --dbname=DBNAME  
  database name
- p PASSWORD, --password=PASSWORD  
  database admin password
- P PACKAGE, --package=PACKAGE  
  specify package or 'all' or 'default'

Usage: upgrade_license_key --database myDB --license my_license.key
upgrade_license_key --install --license my_license.key

Updates the vertica license.

Without '--install', updates the license used by the database and the admintools license cache.

With '--install', updates the license cache in admintools that is used for creating new databases.

Options:
- h, --help  
  show this help message and exit
- d DB, --database=DB  
  Name of database. Cannot be used with --install.
- l LICENSE, --license=LICENSE  
  Required - path to the license.
- i, --install  
  When option is included, command will only update the admintools license cache. Cannot be used with --database.
- p PASSWORD, --password=PASSWORD  
  Database password.

Usage: view_cluster [options]

Options:
- h, --help  
  show this help message and exit
- x, --xpend  
  show the full cluster state, node by node
- d DB, --database=DB  
  filter the output for a single database
Operating the Database

This topic explains how to start and stop your Vertica database, and how to use the database index tool.

Starting the Database

You can start a database through one of the following:

- Management Console
- Administration Tools
- Command Line

Administration Tools

1. Open the Administration Tools and select View Database Cluster State to make sure that all nodes are down and that no other database is running.

2. Open the Administration Tools. See Using the Administration Tools for information about accessing the Administration Tools.

3. On the Main Menu, select Start Database, and then select OK.

4. Select the database to start, and then click OK.

   Caution: You should start only one database at a time. If you start more than one database at any time, the results can be unpredictable. Users might encounter resource conflicts or perform operations on the wrong database.

5. Enter the database password and click OK.

6. When prompted that the database started successfully, click OK.

7. Check the log files to make sure that no startup problems occurred.
Command Line

You can start a database with the command line tool `start_db`:

```
$ /opt/vertica/bin/admintools -t start_db -d db-name
   [-p password] [-F]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-p</code></td>
<td>Required only during database creation, when you install a new license. If the license is valid, the option <code>-p</code> (or <code>--password</code>) is not required to start the database and is silently ignored. This is by design, as the database can only be started by the user who (as part of the verticadba UNIX user group) initially created the database or who has root or su privileges. If the license is invalid, Vertica uses the <code>-p</code> password argument to attempt to upgrade the license with the license file stored in <code>/opt/vertica/config/share/license.key</code>.</td>
</tr>
<tr>
<td><code>-F</code></td>
<td>Forces the database to start at an epoch that precedes detection of data consistency problems</td>
</tr>
</tbody>
</table>

Following is an example of using `start_db` on a standalone node:

```
$ /opt/vertica/bin/admintools -t start_db -d VMart
Info:
   no password specified, using none
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (UP)
Database VMart started successfully
```

Stopping the Database

There are many occasions when you must stop a database, for example, before an upgrade or performing various maintenance tasks. You can stop a running database through one of the following:
You cannot stop a running database if any users are connected or Database Designer is building or deploying a database design.

Note: If the Tuple Mover is doing a Moveout operation then the database cannot stop until the Moveout is complete. If the database is not stopping after you issue a stop command then you can verify a Moveout operation is preventing the database from stopping by looking at the Vertica log file. See Monitoring Log Files for details on locating and viewing the Vertica log. Tuple Mover operations that are in progress display an INFO message: [Session] <INFO> closeAndWaitAllSessions: waiting for session to end. The database will stop after the Moveout completes, you do not need to take additional action other than waiting.

Administration Tools

To stop a running database with admintools:

1. Verify that all cluster nodes are up. If any nodes are down, identify and restart them.

2. Close all user sessions:

   - Identify all users with active sessions by querying the SESSIONS system table. Notify users of the impending shutdown and request them to shut down their sessions.
   
   - Prevent users from starting new sessions by temporarily resetting configuration parameter MaxClientSessions to 0:

     ```sql
     => ALTER DATABASE mydb SET MaxClientSessions = 0;
     ```

   - Close all remaining user sessions with Vertica functions CLOSE_SESSION and CLOSE_ALL_SESSIONS.

   Note: You can also force a database shutdown and block new sessions with the function SHUTDOWN.

3. Open Vertica Administration Tools.
4. From the Main Menu:
   - Select Stop Database
   - Click OK

5. Select the database to stop and click OK.

6. Enter the password (if asked) and click OK.

7. When prompted that database shutdown is complete, click OK.

Command Line

You can stop a database with the command line tool `stop_db`:

```
$ /opt/vertica/bin/admintools -t stop_db -d db-name
    [-p password] [-F]
```

Use the option `-F` (or `--force`) to override all user connections and force a shutdown.
CRC and Sort Order Check

As a superuser, you can run the Index tool on a Vertica database to perform two tasks:

- Run a cyclic redundancy check (CRC) on each block of existing data storage to check the data integrity of ROS data blocks.
- Check that the sort order in ROS containers is correct.

If the database is down, invoke the Index tool from the Linux command line. If the database is up, invoke from VSQL with Vertica metafunction SQL statement `RUN_INDEX_TOOL`:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Database down</th>
<th>Database up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run CRC</td>
<td>/opt/vertica/bin/vertica -D catalog.Catalog -I catalog -F catalog -v <code>RUN_INDEX_TOOL</code>('checkcrc', ...)</td>
<td>/opt/vertica/bin/vertica -D catalog.Catalog -I catalog -F catalog -v <code>RUN_INDEX_TOOL</code>('checkcrc', ...)</td>
</tr>
<tr>
<td>Check sort order</td>
<td>/opt/vertica/bin/vertica -D catalog.Catalog -I catalog -F catalog -v <code>RUN_INDEX_TOOL</code>('checksort', ...)</td>
<td>/opt/vertica/bin/vertica -D catalog.Catalog -I catalog -F catalog -v <code>RUN_INDEX_TOOL</code>('checksort', ...)</td>
</tr>
</tbody>
</table>

If invoked from the command line, the Index tool runs only on the current node. However, you can run the Index tool on multiple nodes simultaneously.

Result Output

The Index tool writes summary information about its operation to standard output; detailed information on results is logged in one of two locations, depending on the environment where you invoke the tool:

<table>
<thead>
<tr>
<th>Invoked from</th>
<th>Results written to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux command line</td>
<td>indextool.log in the database catalog directory</td>
</tr>
<tr>
<td>VSQL</td>
<td>vertica.log on the current node</td>
</tr>
</tbody>
</table>

For information about evaluating output for possible errors, see:

- Evaluating CRC Errors
- Evaluating Sort Order Errors
Optimizing Performance

You can optimize meta-function performance by narrowing the scope of the operation to one or more projections, and specifying the number of threads used to execute the function. For details, see RUN_INDEX_TOOL.

Evaluating CRC Errors

Vertica evaluates the CRC values in each ROS data block each time it fetches data disk to process a query. If CRC errors occur while fetching data, the following information is written to the vertica.log file:

<table>
<thead>
<tr>
<th>CRC Check Failure Details:</th>
<th>File Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Offset:</td>
<td></td>
</tr>
<tr>
<td>Compressed size in file:</td>
<td></td>
</tr>
<tr>
<td>Memory Address of Read Buffer:</td>
<td></td>
</tr>
<tr>
<td>Pointer to Compressed Data:</td>
<td></td>
</tr>
<tr>
<td>Memory Contents:</td>
<td></td>
</tr>
</tbody>
</table>

The Event Manager is also notified of CRC errors, so you can use an SNMP trap to capture CRC errors:

"CRC mismatch detected on file <file_path>. File may be corrupted. Please check hardware and drivers."

If you run a query from vsql, ODBC, or JDBC, the query returns a FileColumnReader ERROR. This message indicates that a specific block's CRC does not match a given record as follows:

| hint: Data file may be corrupt. Ensure that all hardware (disk and memory) is working properly. Possible solutions are to delete the file <pathname> while the node is down, and then allow the node to recover, or truncate the table data.code: ERRCODE_DATA_CORRUPTED |

Evaluating Sort Order Errors

If ROS data is not sorted correctly in the projection's order, query results that rely on sorted data will be incorrect. You can use the Index tool to check the ROS sort order if you suspect or detect incorrect query results. The Index tool evaluates each ROS row to determine whether it is sorted correctly. If the check locates a row that is not in order, it writes an error message to the log file with the row number and contents of the unsorted row.
Reviewing Errors

1. Open the `indextool.log` file. For example:

```
$ cd VMart/v_check_node0001_catalog
```

2. Look for error messages that include an OID number and the string `Sort Order Violation`. For example:

```
<INFO> ... on oid 45035996273723545: Sort Order Violation:
```

3. Find detailed information about the sort order violation string by running `grep` on `indextool.log`. For example, the following command returns the line before each string (-B1), and the four lines that follow (-A4):

```
[15:07:55][vertica-s1]: grep -B1 -A4 'Sort Order Violation:' /my_host/databases/check/v_check_node0001_catalog/indextool.log

2012-06-14 14:07:13.686 unknown:0x7fe1da7a1950 [EE] <INFO> An error occurred when running index tool thread on oid 45035996273723537:
Sort Order Violation:
Row Position: 624
Column Index: 0
Last Row: 2576000
This Row: 2575000
--
2012-06-14 14:07:13.687 unknown:0x7fe1dafa2950 [EE] <INFO> An error occurred when running index tool thread on oid 45035996273723545:
Sort Order Violation:
Row Position: 3
Column Index: 0
Last Row: 4
This Row: 2
--
```

4. Find the projection where a sort order violation occurred by querying system table `STORAGE_CONTAINERS`. Use a `storage_oid` equal to the OID value listed in `indextool.log`. For example:

```
=> SELECT * FROM storage_containers WHERE storage_oid = 45035996273723545;
```
Managing Tables

You can create two types of tables in Vertica, columnar and flexible. You can create both types as persistent or temporary. You can also create views that query a specific set of table columns.

Creating Tables

CREATE TABLE creates a table in the Vertica logical schema. For example:

```sql
CREATE TABLE vendor_dimension (  
  vendor_key INTEGER NOT NULL PRIMARY KEY,  
  vendor_name VARCHAR(64),  
  vendor_address VARCHAR(64),  
  vendor_city VARCHAR(64),  
  vendor_state CHAR(2),  
  vendor_region VARCHAR(32),  
  deal_size INTEGER,  
  lastDealUpdate DATE
);
```

Table Data Storage

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns. Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query.

In order to query or perform any operation on a Vertica table, the table must have one or more projections associated with it. For more information, see Physical Schema in Vertica Concepts.

See Also

- Altering Table Definitions
- Creating Temporary Tables
- Creating a Table from Other Tables
- Creating External Tables
Creating Temporary Tables

`CREATE TEMPORARY TABLE` creates a table whose data persists only during the current session. Temporary table data is never visible to other sessions.

By default, all temporary table data is transaction-scoped—that is, the data is discarded when a `COMMIT` statement ends the current transaction. If `CREATE TEMPORARY TABLE` includes the parameter `ON COMMIT PRESERVE ROWS`, table data is retained until the current session ends.

Temporary tables can be used to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated—for example, the tool first gets a result set, then queries the result set, and so on.

When you create a temporary table, Vertica automatically generates a default projection for it. For more information, see Auto-Projections.

Global versus Local Tables

`CREATE TEMPORARY TABLE` can create tables at two scopes, global and local, through the keywords `GLOBAL` and `LOCAL`, respectively:

| Global temporary tables | Vertica creates global temporary tables in the public schema. Definitions of these tables are visible to all sessions, and persist across sessions until they are explicitly dropped. Multiple users can access the table concurrently. Table data is session-scoped, so it is visible only to the session user, and is discarded when the session ends. |
| Local temporary tables | Vertica creates local temporary tables in the V_TEMP_SCHEMA namespace and inserts them transparently into the user's search path. These tables are visible only to the session where they are created. When the session ends, Vertica automatically drops the table and its data. |

Data Retention

You can specify whether temporary table data is transaction- or session-scoped:
**ON COMMIT DELETE ROWS** (default): Vertica automatically removes all table data when each transaction ends.

**ON COMMIT PRESERVE ROWS**: Vertica preserves table data across transactions in the current session. Vertica automatically truncates the table when the session ends.

**Note**: If you create a temporary table with **ON COMMIT PRESERVE ROWS**, you cannot add projections for that table if it contains data. You must first remove all data from that table with **TRUNCATE TABLE**.

You can create projections for temporary tables created with **ON COMMIT DELETE ROWS**, whether populated with data or not. However, **CREATE PROJECTION** ends any transaction where you might have added data, so projections are always empty.

**ON COMMIT DELETE ROWS**
By default, Vertica removes all data from a temporary table, whether global or local, when the current transaction ends.

For example:

```sql
=> CREATE TEMPORARY TABLE tempDelete (a int, b int);
CREATE TABLE
=> INSERT INTO tempDelete VALUES(1,2);
    OUTPUT --------
        1
(1 row)

=> SELECT * FROM tempDelete;
a | b
---+---
1 | 2
(1 row)

=> COMMIT;
COMMIT

=> SELECT * FROM tempDelete;
a | b
---+---
(0 rows)
```

If desired, you can use **DELETE** within the same transaction multiple times, in order to refresh table data repeatedly.

**ON COMMIT PRESERVE ROWS**
You can specify that a temporary table retain data across transactions in the current session, by defining the table with the keywords **ON COMMIT PRESERVE ROWS**. Vertica automatically removes all data from the table only when the current session ends.

For example:
Creating a Table from Other Tables

You can create a table from other tables in two ways:

- **Replicate an existing table** through `CREATE TABLE ... LIKE`.
- **Create a table from a query** through `CREATE TABLE ... AS`.

Replicating a Table

You can create a table from an existing one using `CREATE TABLE` with the LIKE clause:

```
CREATE TABLE [schema.]table-name LIKE [schema.]existing-table 
...[[INCLUDING | EXCLUDING] PROJECTIONS ]
...[ load-method ]
...[[INCLUDE | EXCLUDE] [SCHEMA] PRIVILEGES ]
```
Creating a table with LIKE replicates the source table definition and any storage policy associated with it. It does not copy table data or expressions on columns.

Copying Constraints

CREATE TABLE...LIKE copies all table constraints, with the following exceptions:

- Foreign key constraints.
- Any column that obtains its values from a sequence, including IDENTITY and AUTO_INCREMENT columns. Vertica copies the column values into the new table, but removes the original constraint. For example, the following table definition sets an IDENTITY constraint on column ID:

```sql
CREATE TABLE public.Premium_Customer
(
   ID IDENTITY,
   lname varchar(25),
   fname varchar(25),
   store_membership_card int
);
```

The following CREATE TABLE...LIKE statement replicates this table as All_Customers. Vertica removes the IDENTITY constraint from All_Customers.ID, changing it to an integer column with a NOT NULL constraint:

```sql
=> CREATE TABLE All_Customers like Premium_Customer;
CREATE TABLE
=> select export_tables('','All_Customers');
   export_tables
-----------------------------------------------
CREATE TABLE public.All_Customers
(
   ID int NOT NULL,
   lname varchar(25),
   fname varchar(25),
   store_membership_card int
);
(1 row)
```

Including Projections

You can qualify the LIKE clause with INCLUDING PROJECTIONS or EXCLUDING PROJECTIONS, which specify whether to copy projections from the source table:
• EXCLUDING PROJECTIONS (default): Do not copy projections from the source table.

• INCLUDING PROJECTIONS: Copy current projections from the source table. Vertica names the new projections according to Vertica naming conventions, to avoid name conflicts with existing objects.

Specifying a Load Method

You can qualify the LIKE clause with a load method, one of the following:

• AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.

• DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.

• TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.

For details, see Choosing a Load Method in the Administrator's Guide.

Including Schema Privileges

You can specify default inheritance of schema privileges for the new table:

• EXCLUDE [SCHEMA] PRIVILEGES (default) disables inheritance of privileges from the schema

• INCLUDE [SCHEMA] PRIVILEGES grants the table the same privileges granted to its schema

For more information see Grant Inherited Privileges.

Restrictions

The following restrictions apply to the source table:

• It cannot have out-of-date projections.

• It cannot be a temporary table.
Example

1. Create the table states:

```sql
=> CREATE TABLE states (
    state char(2) NOT NULL, bird varchar(20), tree varchar(20), tax float, stateDate char(20)
) PARTITION BY state;
```

2. Populate the table with data:

```sql
INSERT INTO states VALUES ('MA', 'chickadee', 'american_elm', 5.675, '07-04-1620');
INSERT INTO states VALUES ('VT', 'Hermit Thrasher', 'Sugar_Maple', 6.0, '07-04-1610');
INSERT INTO states VALUES ('NH', 'Purple Finch', 'White_Birch', 0, '07-04-1615');
INSERT INTO states VALUES ('ME', 'Black_Cap_Chickadee', 'Pine_Tree', 5, '07-04-1615');
INSERT INTO states VALUES ('CT', 'American Robin', 'White_Oak', 6.35, '07-04-1618');
INSERT INTO states VALUES ('RI', 'Rhode_Island_Red', 'Red_Maple', 5, '07-04-1619');
```

3. View the table contents:

```sql
=> SELECT * FROM states;
```

<table>
<thead>
<tr>
<th>state</th>
<th>bird</th>
<th>tree</th>
<th>tax</th>
<th>stateDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT</td>
<td>Hermit Thrasher</td>
<td>Sugar_Maple</td>
<td>6</td>
<td>07-04-1610</td>
</tr>
<tr>
<td>CT</td>
<td>American Robin</td>
<td>White_Oak</td>
<td>6.35</td>
<td>07-04-1618</td>
</tr>
<tr>
<td>RI</td>
<td>Rhode_Island_Red</td>
<td>Red_Maple</td>
<td>5</td>
<td>07-04-1619</td>
</tr>
<tr>
<td>MA</td>
<td>chickadee</td>
<td>american_elm</td>
<td>5.675</td>
<td>07-04-1620</td>
</tr>
<tr>
<td>NH</td>
<td>Purple Finch</td>
<td>White_Birch</td>
<td>0</td>
<td>07-04-1615</td>
</tr>
<tr>
<td>ME</td>
<td>Black_Cap_Chickadee</td>
<td>Pine_Tree</td>
<td>5</td>
<td>07-04-1615</td>
</tr>
</tbody>
</table>

(6 rows)

4. Create a sample projection and refresh:

```sql
=> CREATE PROJECTION states_p AS SELECT state FROM states;
=> SELECT START_REFRESH();
```

5. Create a table like the states table and include its projections:

```sql
=> CREATE TABLE newstates LIKE states INCLUDING PROJECTIONS;
```

6. View projections for the two tables. Vertica has copied projections from states to newstates:
=> \dj

<table>
<thead>
<tr>
<th>Node</th>
<th>Schema</th>
<th>Comment</th>
<th>Name</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>newstates_b0</td>
<td></td>
<td>newstates_b0</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>newstates_b1</td>
<td></td>
<td>newstates_b1</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>newstates_p_b0</td>
<td></td>
<td>newstates_p_b0</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>newstates_p_b1</td>
<td></td>
<td>newstates_p_b1</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>states_b0</td>
<td></td>
<td>states_b0</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>states_b1</td>
<td></td>
<td>states_b1</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>states_p_b0</td>
<td></td>
<td>states_p_b0</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>states_p_b1</td>
<td></td>
<td>states_p_b1</td>
<td>dbadmin</td>
</tr>
</tbody>
</table>

7. View the table newstates, which shows columns copied from states:

=> SELECT * FROM newstates;

state | bird | tree | tax | stateDate
-----------------------------
(0 rows)

When you use the CREATE TABLE...LIKE statement, storage policy objects associated with the table are also copied. Data added to the new table use the same labeled storage location as the source table, unless you change the storage policy. For more information, see Working With Storage Locations.

See Also

- Creating Tables
- Creating Temporary Tables
- Creating External Tables
- Creating a Table from a Query
Creating a Table from a Query

`CREATE TABLE` can specify an AS clause, to create a table from a query, as follows:

```
CREATE [TEMPORARY] TABLE [schema.]table-name
  ... [(column-name-list)]
  ... [load-method]
  ... [{INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES]
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list]
```

Vertica creates a table from the query results and loads the result set into it. For example:

```
=> CREATE TABLE cust_basic_profile AS SELECT
    customer_key, customer_gender, customer_age, marital_status, annual_income, occupation
  FROM customer_dimension WHERE customer_age>18 AND customer_gender !='';
CREATE TABLE
=> SELECT customer_age, annual_income, occupation FROM cust_basic_profile
  WHERE customer_age > 23 ORDER BY customer_age;
```

AS Clause Options

You can qualify the AS clause with one or both of the following hints:

- A load method hint: **AUTO**, **DIRECT**, or **TRICKLE**

  **Note:** The `CREATE TABLE` statement can also specify a load method. However, this load method applies to load operations only after the table is created.

- **LABEL**: Assigns a label to a statement to identify it for profiling and debugging. See Labeling the AS Clause below.

You can also specify that the query return historical data with **AT epoch**, where `epoch` is one of the following:

- **EPOCH LATEST**: Return data from the latest committed DML transaction.
- **EPOCH integer**: Return data from the `integer`-specified epoch.
TIME 'timestamp': Return data from the timestamp-specified epoch.

Note: These options are invalid for external tables.

Labeling the AS Clause

You can embed a LABEL hint in an AS clause in two places:

- Immediately after the keyword AS:

  CREATE TABLE myTable AS /*+LABEL myLabel*/...

- In the SELECT statement:

  CREATE TABLE myTable AS SELECT /*+LABEL myLabel*/

If the AS clause contains a LABEL hint in both places, the first label has precedence.

Note: Labels are invalid for external tables.

Specifying a Load Method

You can qualify the AS clause with one of these load method hints:

- AUTO
- DIRECT
- TRICKLE

Vertica applies the load hint to the data that is loaded into the new table. You can also specify a load method in the CREATE TABLE statement, which is saved to the table schema. However, the load method applies only to load operations after the table is created.

In the following example, table bar is created and loaded from table foo with a CREATE TABLE AS statement. The CREATE statement is qualified with the load method DIRECT, which is saved to bar's schema definition. The load method specifies that all future load operations are written to ROS. The CREATE statement's AS clause also contains the load method hint /*+DIRECT*/, which specifies to load the data queried from foo into ROS:
=> SELECT * FROM foo;
  col1 | col2 | col3
  -------+-------+-------
   4    |   5   |   6   
   1    |   2   |   3   
(2 rows)

=> CREATE TABLE bar DIRECT AS /*+DIRECT*/ * FROM foo;
CREATE TABLE dbadmin=> SELECT EXPORT_TABLES ('','bar');
EXPORT_TABLES
  -------+-------+-------
   4    |   5   |   6   
   1    |   2   |   3   
(2 rows)

CREATE TABLE public.bar
  (col1 int,   
   col2 int,   
   col3 int
  ) DIRECT ;
(1 row)

For details, see Choosing a Load Method.

Note: Load methods cannot be specified for external tables.

Loading Historical Data

You can qualify a CREATE TABLE AS query with the option, with AT epoch. AT epoch specifies to return historical data, where epoch is one of the following:

- **EPOCH LATEST**: Return data from the latest committed DML transaction.
- **EPOCH integer**: Return data from the integer-specified epoch.
- **TIME 'timestamp'**: Return data from the timestamp-specified epoch.

For details, see Historical Queries in Analyzing Data.

Note: This option is invalid for external tables.

Zero-Width Column Handling

If the query returns a column with zero width, Vertica automatically converts it to a VARCHAR (80) column. For example:

=> CREATE TABLE example AS SELECT '' AS X;
CREATE TABLE
=> SELECT EXPORT_TABLES ('', 'example');
Requirements and Restrictions

- If you create a temporary table from a query, you must specify `ON COMMIT PRESERVE ROWS` in order to load the result set into the table. Otherwise, Vertica creates an empty table.

- If the query output has expressions other than simple columns, such as constants or functions, you must specify an alias for that expression, or list all columns in the column name list.

See Also

- Creating Tables
- Creating Temporary Tables
- Creating External Tables
- Replicating a Table

Creating External Tables

You create an external table using the `CREATE EXTERNAL TABLE AS COPY` statement. You cannot create temporary external tables. For the syntax details to create an external table, see the `CREATE EXTERNAL TABLE AS COPY` statement in the SQL Reference Manual.

Note: Each table can have a maximum of 1600 columns.

Required Permissions for External Tables

You must be a database superuser to create external tables.
Permission requirements to use (SELECT from) external tables differ from those of other tables. By default, once external tables exist, you must also be a database superuser to access them through a SELECT statement.

To allow users without superuser access to query external tables, an administrator must create a 'user' storage location and grant those users read access to the location. See CREATE LOCATION, and GRANT (Storage Location). This location must be a parent of the path used in the COPY statement when creating the external table.

COPY Statement Definition

When you create an external table, table data is not added to the database, and no projections are created. Instead, Vertica performs a syntactic check of the CREATE EXTERNAL TABLE... statement, and stores the table name and COPY statement definition in the catalog. Each time a SELECT query references an external table, Vertica parses and executes the stored COPY statement to obtain the referenced data. Successfully returning data from an external table requires that the COPY definition be correct, and that other dependencies, such as files, nodes, and other resources are accessible and available at query time. If the table definition uses globs (wildcards), and files are added or deleted, the data in the external table can change between queries.

If the maximum length of a column is smaller than the actual data, such as a VARCHAR that is too short, Vertica truncates the data and logs the event.

If your data is in Parquet or ORC format, you can take advantage of partitioning to limit the amount of data that Vertica reads. See Using Partition Columns.

When using the COPY parameter on any node, confirm that the source file definition is identical on all nodes. Specifying different external files can produce inconsistent results.

For more information about checking the validity of the external table COPY definition, see Validating External Tables.

NOT NULL Constraints

Do not specify a NOT NULL column constraint, unless you are certain that the external data does not contain NULL values. Otherwise, you may see unexpected query results. For example, a SELECT statement for an external table with a NOT NULL constraint will reject a column value if it is not NULL.
Canceling the Create Query

Canceling a CREATE EXTERNAL TABLE AS COPY statement can cause unpredictable results. If you enter a query to create an external table, and it is incorrect (for example, you inadvertently specify the wrong external location), wait for the query to complete. When the external table exists, use DROP TABLEs definition.

Developing User-Defined Load (UDL) Functions for External Tables

You can create external tables with your own load functions. For more information about developing user-defined load functions, see User Defined Load (UDL) and the extended COPY syntax in the SQL Reference Manual.

Examples

Examples of external table definitions:

```
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat' DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM 'hdfs:///dat/ext1.dat';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat.bz2' BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE ext2 (x integer, y integer) AS COPY (x as '5', y) FROM '/tmp/ext1.dat.bz2'
    BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE sales (itemID INT, date DATE, price FLOAT)
    AS COPY FROM 's3://datalake/sales/*.parquet' PARQUET;
```

To allow users without superuser access to use these tables, create a location for 'user' usage and grant access to it. This example shows granting access to a user named Bob to any external table whose data is located under /tmp (including in subdirectories to any depth):

```
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';
=> GRANT ALL ON LOCATION '/tmp' to Bob;
```

See Also

- COPY
- CREATE EXTERNAL TABLE AS COPY
Validating External Tables

When you create an external table, Vertica validates the syntax of the CREATE EXTERNAL TABLE AS COPY FROM statement. For example, if you omit a required keyword in the statement (such as FROM), creating the external table fails:

```sql
VMart=> create external table ext (ts timestamp,d varchar) as copy '/home/dbadmin/designer.log';
ERROR 2778: COPY requires a data source; either a FROM clause or a WITH SOURCE for a user-defined source
```

Checking other components of the COPY definition (such as path statements and node availability) does not occur until a SELECT query references the external table.

To validate an external table definition, run a SELECT query that references the external table. Check that the returned query data is what you expect. If the query does not return data correctly, check the COPY exception and rejected data log files.

Since the COPY definition determines what occurs when you query an external table, COPY statement errors can reveal underlying problems. For more information about COPY exceptions and rejections, see Capturing Load Rejections and Exceptions.

Setting Maximum Exceptions

Querying external table data with an incorrect COPY FROM statement definition can potentially result in many rejected rows. To limit the number of rejections, Vertica sets the maximum number of retained rejections with the `ExternalTablesExceptionsLimit` configuration parameter. The default value is 100. Setting the `ExternalTablesExceptionsLimit` to -1 removes the limit, but is not recommended.

If COPY errors reach the maximum number of rejections, the external table query continues, but COPY generates a warning in the `vertica.log`, and does not report subsequent rejected rows.

```
Note: Using the `ExternalTablesExceptionsLimit` configuration parameter differs from the COPY statement `REJECTMAX` parameter. The `REJECTMAX` value controls how many rejected rows to permit before causing the load to fail. If COPY encounters a number of rejected rows equal to or greater than `REJECTMAX`, COPY aborts execution. A `vertica.log` warning is not generated if COPY exceeds `REJECTMAX`.
```
Working with External Tables

After creating external tables, you access them as any other table. However, you cannot perform UPDATE, INSERT, or DELETE operations on external tables.

Managing Resources for External Tables

External tables require minimal additional resources. When you use a select query for an external table, Vertica uses a small amount of memory when reading external table data, since the table contents are not part of your database and are parsed each time the external table is used.

Vertica Does Not Back Up External Tables

Since the data in external tables is managed outside of Vertica, only the external table definitions, not the data files, are included in database backups. Arrange for a separate backup process for your external table data.

Using Sequences and Identity Columns in External Tables

The COPY statement definition for external tables can include identity columns and sequences. Whenever a select statement queries the external table, sequences and identity columns are re-evaluated. This results in changing the external table column values, even if the underlying external table data remains the same.

Viewing External Table Definitions

When you create an external table, Vertica stores the COPY definition statement in the table_definition column of the v_catalog.tables system table.

1. To list all tables, use a select * query, as shown:

   ```sql
   select * from v_catalog.tables where table_definition <> '';
   ```
2. Use a query such as the following to list the external table definitions (table_definition):

```sql
select table_name, table_definition from v_catalog.tables;
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>COPY FROM 'TMPDIR/external_table.dat' DELIMITER ','</td>
</tr>
<tr>
<td>t1_copy</td>
<td>COPY FROM 'TMPDIR/external_table.dat' DELIMITER ','</td>
</tr>
<tr>
<td>t2</td>
<td>COPY FROM 'TMPDIR/external_table2.dat' DELIMITER ','</td>
</tr>
</tbody>
</table>

(3 rows)

**External Table DML Support**

Following are examples of supported queries, and others that are not:

<table>
<thead>
<tr>
<th>Supported</th>
<th>Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM external_table;</td>
<td>DELETE FROM external_table WHERE x = 5;</td>
</tr>
<tr>
<td>SELECT * FROM external_table where col1=4;</td>
<td>INSERT INTO external_table SELECT * FROM ext;</td>
</tr>
<tr>
<td>DELETE FROM internal_table WHERE id IN (SELECT x FROM external_table);</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO internal_table SELECT * FROM external_table;</td>
<td>SELECT * FROM external_table FOR UPDATE;</td>
</tr>
</tbody>
</table>

**Using External Table Values**

Following is a basic example of how you could use the values of an external table.

1. Create and display the contents of a file with some integer values:

```
[dbadmin@localhost ~]$ more ext.dat1
2
3
4
5
6
7
8
10
11
12
```
2. Create an external table pointing at ext.dat:

VMart=> create external table ext (x integer) as copy from '/home/dbadmin/ext.dat';

CREATE TABLE

3. Select the table contents:

VMart=> select * from ext;
  x
  ----
  1
  2
  3
  4
  5
  6
  7
  8
  10
  11
  12
(11 rows)

4. Perform evaluation on some external table contents:

VMart=> select ext.x, ext.x + ext.x as double_x from ext where x > 5;
  x | double_x
  ---- |---------
  6 | 12
  7 | 14
  8 | 16
 10 | 20
 11 | 22
 12 | 24
(6 rows)

5. Create a second table (second), also with integer values:

VMart=> create table second (y integer);
CREATE TABLE

6. Populate the table with some values:

VMart=> copy second from stdin;Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
  >> 1
  >> 1
  >> 3
  >> 4
7. Join the external table (ext) with the table created in Vertica, called second:

```
VMart=> select * from ext join second on x=y;
 x | y 
---+---
 1 | 1
 1 | 1
 3 | 3
 4 | 4
 5 | 5
(5 rows)
```

## Using External Tables

External tables let you query data stored in files accessible to the Vertica database, but not managed by it. Creating external tables supplies read-only access through SELECT queries. You cannot modify external tables through DML commands, such as INSERT, UPDATE, DELETE, and MERGE.

### Using CREATE EXTERNAL TABLE AS COPY Statement

You create external tables with the `CREATE EXTERNAL TABLE AS COPY...` statement, shown in this basic example:

```
CREATE EXTERNAL TABLE tbl(i INT) AS COPY (i) FROM 'path1' ON v_vmart_node0001, 'path2' ON v_vmart_node0002;
```

For more details on the supported options to create an external table, see the `CREATE EXTERNAL TABLE` statement in the SQL Reference Manual.

The data you specify in the `FROM` clause of a `CREATE EXTERNAL TABLE AS COPY` statement can reside in one or more files or directories, and on one or more nodes. After successfully creating an external table, Vertica stores the table name and its COPY definition. Each time a select query references the external table, Vertica parses the COPY statement definition again to access the data. Here is a sample select statement:

```
SELECT * FROM tbl WHERE i > 10;
```
Storing Vertica Data in External Tables

While there are many requirements for you to use external table data, one reason is to store infrequently-accessed Vertica data on low-cost external media. If external storage is a goal at your site, the process to accomplish that requires exporting the older data to a text file, creating a bzip or gzip file of the export data, and saving the compressed file on an NFS disk. You can then create an external table to access the data any time it is required.

Calculating Exact Row Count for External Tables

To calculate the exact number of rows in an external table, use `ANALYZE_EXTERNAL_ROW_COUNT`. The Optimizer uses this count to optimize for queries that access external tables.

In particular, if an external table participates in a join, the Optimizer can now identify the smaller table to be used as the inner input to the join, resulting in better query performance.

Using External Tables with User-Defined Load (UDL) Functions

You can also use external tables in conjunction with the UDL functions that you create. For more information about using UDLs, see User Defined Load (UDL) in Extending Vertica.

Organizing External Table Data

If the data you store in external tables changes regularly (for instance, each month in the case of storing recent historical data), your COPY definition statement can use wildcards to make parsing the stored COPY statement definition more dynamic. For instance, if you store monthly data on an NFS mount, you could organize monthly files within a top-level directory for a calendar year, such as:

```
/2012/monthly_archived_data/
```

In this case, the external table COPY statement will include a wildcard definition such as the following:

```
CREATE TABLE archive_data (...) AS COPY FROM '/nfs_name/2012/monthly_archived_data/*'
```
Whenever a Vertica query references the external table months, and Vertica parses the COPY statement, all stored data tables in the top-level monthly_archived_data directory are made accessible to the query.

### Managing Table Columns

After you define a table, you can use `ALTER TABLE` to modify existing table columns. You can perform the following operations on a column:

- **Rename it.**
- **Change its data type.**
- **Set its default value.**
- **Add and remove** constraints.

#### Renaming Columns

You rename a column with `ALTER TABLE` as follows:

```
ALTER TABLE [schema.]table-name RENAME [ COLUMN ] column-name TO new-column-name
```

The following example renames a column in the Retail.Product_Dimension table from `Product_description` to `Item_description`:

```
=> ALTER TABLE Retail.Product_Dimension RENAME COLUMN Product_description TO Item_description;
```

If you rename a column that is referenced by a view, the column does not appear in the result set of the view even if the view uses the wild card (*) to represent all columns in the table. Recreate the view to incorporate the column's new name.

#### Changing a Column Data Type

You can change a table column's data type with `ALTER TABLE` as long as the change complies with the requirements and restrictions cited below.
Supported Data Type Conversions

You can change a column's data type if doing so does not require storage reorganization. After you modify a column's data type, data that you load conforms to the new definition.

Vertica supports conversion between the following data types:

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary types</td>
<td>Expansion and contraction allowed. Conversion not allowed between BINARY and VARBINARY types.</td>
</tr>
<tr>
<td>Character types</td>
<td>All conversions allowed, including between CHAR and VARCHAR</td>
</tr>
<tr>
<td>Exact numeric types</td>
<td>INTEGER, INT, BIGINT, TINYINT, INT8, SMALLINT, and all NUMERIC values of scale &lt;=18 and precision 0 are interchangeable. For NUMERIC data types, you cannot alter scale, but you can change the precision in the ranges (0-18), (19-37), and so on.</td>
</tr>
</tbody>
</table>

You can expand columns within the same class of data type. This is useful for storing longer strings in a column. Vertica validates the data before it performs the conversion. In general, you can also reduce column widths within the data type class, as long as existing column data is no greater than the new width. For details, see Reducing Column Width.

Unsupported Data Type Conversions

Vertica does not allow data type conversion on types that require storage reorganization:

- Boolean type conversion to other types
- DATE/TIME type conversion
- Approximate numeric type conversions
- Between BINARY and VARBINARY types

You also cannot change a column's data type if the column is one of the following:

- Primary key
- Foreign key
• Included in the SEGMENTED BY clause of any projection for that table.

You can work around some of these restrictions. For details, see Working With Column Data Conversions.

Reducing Column Width

In general, Vertica supports reductions to column widths within the data type class, as long as existing column data is no greater than the new width. Otherwise, Vertica returns an error and the conversion fails. For example, if you try to convert a column from varchar(25) to varchar(10) Vertica allows the conversion as long as all column data is no more than 10 characters.

In the following example, columns y and z are initially defined as VARCHAR data types, and loaded with values 12345 and 654321, respectively. The attempt to reduce column z's width to 5 fails because it contains six-character data. The attempt to reduce column y's width to 5 succeeds because its content conforms with the new width:

```
=> CREATE TABLE t (x int, y VARCHAR, z VARCHAR);
CREATE TABLE
=> CREATE PROJECTION t_p1 AS SELECT * FROM t SEGMENTED BY hash(x) ALL NODES;
CREATE PROJECTION
=> INSERT INTO t values(1,'12345','654321');
OUTPUT
--------
1
(1 row)

=> SELECT * FROM t;
 x | y    | z
---+-------+-----
 1 | 12345 | 654321
(1 row)

=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ROLLBACK 2378: Cannot convert column "z" to type "char(5)"
HINT: Verify that the data in the column conforms to the new type
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE char(5);
ALTER TABLE
```

Purging Historical Data

You cannot reduce a column's width if Vertica retains any historical data that exceeds the new width. To reduce the column width, first remove that data from the table:
1. Advance the AHM to an epoch more recent than the historical data that needs to be removed from the table.

2. Purge the table of all historical data that precedes the AHM with the function `PURGE_TABLE`.

For example, given the previous example, you can update the data in column `t.z` as follows:

```sql
=> UPDATE t SET z = '54321';
OUTPUT
--------
  1
(1 row)
```

```sql
=> SELECT * FROM t;
  x | y | z
---------
  1 | 12345 | 54321
(1 row)
```

Although no data in column `z` now exceeds 5 characters, Vertica retains the history of its earlier data, so attempts to reduce the column width to 5 return an error:

```sql
=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ROLLBACK 2378: Cannot convert column "z" to type "char(5)"
HINT: Verify that the data in the column conforms to the new type
```

You can reduce the column width by purging the table's historical data as follows:

```sql
=> SELECT MAKE_AHM_NOW();
  MAKE_AHM_NOW
---------------------
   AHM set (New AHM Epoch: 6350)
(1 row)
```

```sql
=> SELECT PURGE_TABLE('t');
PURGE_TABLE
-----------------
Task: purge operation
(Table: public.t) (Projection: public.t_p1_b0)
(Table: public.t) (Projection: public.t_p1_b1)
(1 row)
```

```sql
=> ALTER TABLE t ALTER COLUMN z SET DATA TYPE char(5);
ALTER TABLE
```

Working With Column Data Conversions

Vertica conforms to the SQL standard by disallowing certain data conversions for table columns. However, you sometimes need to work around this restriction when you convert data
from a non-SQL database. The following examples describe one such workaround, using the following table:

```
=> CREATE TABLE sales(id INT, price VARCHAR) UNSEGMENTED ALL NODES;
CREATE TABLE
=> INSERT INTO sales VALUES (1, '$50.00');
OUTPUT
--------
 1
(1 row)
=> INSERT INTO sales VALUES (2, '$100.00');
OUTPUT
--------
 1
(1 row)
=> COMMIT;
COMMIT
=> SELECT * FROM sales;
id | price
--------
1 | $50.00
2 | $100.00
(2 rows)
```

To convert the `price` column's existing data type from VARCHAR to NUMERIC, complete these steps:

1. Add a new column for temporary use. Assign the column a NUMERIC data type, and derive its default value from the existing `price` column.
2. Drop the original `price` column.
3. Rename the new column to the original column.

Add a new column for temporary use

1. Add a column `temp_price` to table `sales`. You can use the new column temporarily, setting its data type to what you want (NUMERIC), and deriving its default value from the `price` column. Cast the default value for the new column to a NUMERIC data type and query the table:

```
=> ALTER TABLE sales ADD COLUMN temp_price NUMERIC(10,2) DEFAULT SUBSTR(sales.price, 2)::NUMERIC;
ALTER TABLE
=> SELECT * FROM sales;
id | price | temp_price
```
2. Use `ALTER TABLE` to drop the default expression from the new column `temp_price`. Vertica retains the values stored in this column:

```sql
=> ALTER TABLE sales ALTER COLUMN temp_price DROP DEFAULT;
ALTER TABLE
```

Drop the original price column

Drop the extraneous `price` column. Before doing so, you must first advance the AHM to purge historical data that would otherwise prevent the drop operation:

1. Advance the AHM:

```sql
=> SELECT MAKE_AHM_NOW();
MAKE_AHM_NOW

----------------------------------
AHM set (New AHM Epoch: 6354)
(1 row)
```

2. Drop the original price column:

```sql
=> ALTER TABLE sales DROP COLUMN price CASCADE;
ALTER COLUMN
```

Rename the new column to the original column

You can now rename the `temp_price` column to `price`:

1. Use `ALTER TABLE` to rename the column:

```sql
=> ALTER TABLE sales RENAME COLUMN temp_price to price;
```

2. Query the sales table again:

```sql
=> SELECT * FROM sales;
   id | price
   ----------
    1 | 50.00
```
Changing Column Types in External Tables

Data from external tables is not stored in Vertica, so Vertica cannot validate any changes that you make in column data types. Vertica checks external table definitions only when it tries to read table data. If external column data is incompatible with the data type that is declared in the Vertica table, Vertica returns an error.

For example:

```sql
=> CREATE EXTERNAL TABLE t (a char(10), b binary(20)) AS COPY FROM '...';
=> ALTER TABLE t ALTER COLUMN a SET DATA TYPE long varchar(1000000);
=) ALTER TABLE t ALTER COLUMN b SET DATA TYPE long varbinary(1000000);
=) ALTER TABLE
```

If you convert a column to a size that is too small for the data, Vertica truncates the data during the read. For example, if you convert a column from `varchar(25)` to `varchar(10)` and the column holds a 20-character string, Vertica reads the first ten and logs a truncation event.

Defining Column Values

You can define a column so Vertica automatically sets its value from an expression through one of the following clauses:

<table>
<thead>
<tr>
<th>DEFAULT expression</th>
<th>Sets the default column values immediately in the following cases:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Adding a column with a DEFAULT expression to an existing table. Vertica populates the new column with its default values when it is added to the table.</td>
</tr>
</tbody>
</table>

Note: Altering an existing table column to specify a DEFAULT expression has no effect on existing values in that column. Vertica applies the DEFAULT expression only on new rows when they are added to the table, through load operations such as INSERT and COPY. To refresh the
entire column with its DEFAULT expression, update the column as follows:

```sql
UPDATE table-name SET column-name=DEFAULT;
```

- Executing a table load operation such as `INSERT`. Vertica populates columns with default values in the new rows. Values in existing rows, including columns with DEFAULT expressions, remain unchanged.

- Executing `UPDATE` on a table and setting default column to DEFAULT. See the example below.

| SET USING expression | Sets the column value only when the function `REFRESH_COLUMNS` is invoked. This approach is especially useful for large denormalized (flattened) tables, where multiple columns get their values by querying other tables. For details, see Flattened Tables.
In general, `REFRESH_COLUMNS` must be explicitly invoked by the user. One exception applies: when you use `ALTER TABLE...ALTER COLUMN` to apply `SET USING` to an existing column, or modify an existing `SET USING` expression. In this case, the DDL operation automatically invokes `REFRESH_COLUMNS` on the column, using UPDATE mode. After the refresh operation is complete, the DDL operation auto-commits the updates and returns. If the refresh operation fails, Vertica rolls back the entire DDL operation. Execution time can be significant if the refresh operation involves a large data set. |

### Supported Expressions

`DEFAULT` and `SET USING` generally support the same expressions. These include:

- Queries (see Flattened Tables)
- Other columns in the same table
- Literals (constants)
- All operators supported by Vertica
The following categories of functions:

- Null-handling
- User-defined scalar
- System information
- String
- Mathematical
- Formatting

Expression Restrictions

The following restrictions apply to DEFAULT and SET USING expressions:

- The return value data type must match or be cast to the column data type.
- The expression must return a value that conforms to the column bounds. For example, a column that is defined as a VARCHAR(1) cannot be set to a default string of abc.
- The expression cannot specify correlated sub-queries.
- In a temporary table, DEFAULT and SET USING do not support sub-queries. If you try to create a temporary table with DEFAULT or SET USING using subquery expressions, Vertica returns an error.
- A column's SET USING expression cannot specify another column in the same table that also sets its value with SET USING. Similarly, a column's DEFAULT expression cannot specify another column in the same table that also sets its value with DEFAULT, or whose value is automatically set to a sequence. However, a column's SET USING expression can specify another column that sets its value with DEFAULT.

Note: You can set a column's DEFAULT expression from another column in the same table that sets its value with SET USING. However, the DEFAULT column is typically set to NULL, as it is only set on load operations that initially set the SET USING column to NULL.

DEFAULT Restrictions

DEFAULT expressions cannot specify volatile functions with ALTER TABLE...ADD COLUMN. To specify volatile functions, use CREATE TABLE or ALTER TABLE...ALTER COLUMN statements.
SET USING Restrictions

The following restrictions apply to SET USING expressions:

- Volatile functions are not allowed.
- The expression cannot specify a sequence.
- You cannot modify SET USING data directly with DML operations such as COPY, INSERT, UPDATE, or MERGE. SET USING columns can only be updated by calling REFRESH_COLUMNS. Attempts to modify SET USING columns return an error.
- Vertica limits the use of several meta-functions that copy table data: COPY_TABLE, COPY_PARTITIONS_TO_TABLE, MOVE_PARTITIONS_TO_TABLE, and SWAP_PARTITIONS_BETWEEN_TABLES. The following table describes these limitations:

<table>
<thead>
<tr>
<th>SET USING columns in...</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source and target table</td>
<td>All functions allowed only if each SET USING column in source table has a corresponding SET USING column in target table.</td>
</tr>
<tr>
<td>Source table only</td>
<td>Allowed functions:</td>
</tr>
<tr>
<td></td>
<td>■ COPY_TABLE</td>
</tr>
<tr>
<td></td>
<td>■ COPY_PARTITIONS_TO_TABLE</td>
</tr>
<tr>
<td></td>
<td>■ MOVE_PARTITIONS_TO_TABLE</td>
</tr>
<tr>
<td></td>
<td>Disallowed: SWAP_PARTITIONS_BETWEEN_TABLE</td>
</tr>
<tr>
<td></td>
<td>The same set of restrictions apply if the target table is new.</td>
</tr>
<tr>
<td>Target table only</td>
<td>No functions allowed.</td>
</tr>
</tbody>
</table>

Important: Several restrictions apply to Vertica's ability to refresh a SET USING column with REFRESH_COLUMNS. For details, see REFRESH_COLUMNS.

DEFAULT USING Columns and Restrictions

A column can specify both DEFAULT and SET USING constraints, as follows:

```
column-name data-type DEFAULT default-expr SET USING using-expr
```
Typically, both constraints specify the same expression. In this case, you can define the column as follows:

\texttt{column-name data-type DEFAULT USING expression}

DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same restrictions.

**Examples**

Derive a column's default value from another column

1. Create table \( t \) with two columns, date and state, and insert a row of data:

\[
=> \text{CREATE TABLE } t \text{ (date DATE, state VARCHAR(2));}
=> \text{CREATE TABLE}
=> \text{INSERT INTO } t \text{ VALUES (CURRENT_DATE, 'MA');}
\]

\[
\text{OUTPUT}
\]

\[
\text{---------}
\]

\[
1
\]

\[
\text{(1 row)}
\]

\[
=> \text{COMMIT;}
\]

\[
\text{COMMIT}
\]

\[
\text{SELECT * FROM } t;
\]

\[
\text{date | state}
\]

\[
\text{---------+---------}
\]

\[
2017-12-28 | MA
\]

\[
\text{(1 row)}
\]

2. Use ALTER TABLE to add a third column that extracts the integer month value from column date:

\[
=> \text{ALTER TABLE } t \text{ ADD COLUMN month INTEGER DEFAULT date_part('month', date);} \]

ALTER TABLE

3. When you query table \( t \), Vertica returns the number of the month in column date:

\[
=> \text{SELECT * FROM } t;
\]

\[
\text{date | state | month}
\]

\[
\text{---------+---------+-------}
\]

\[
2017-12-28 | MA | 12
\]

\[
\text{(1 row)}
\]
Update default column values

1. Update table t by subtracting 30 days from date:

   => UPDATE t SET date = date-30;
   OUTPUT
   -------
     1
   (1 row)

   => COMMIT;
   COMMIT

   => SELECT * FROM t;
   date  | state | month
   -------+-------+------
     2017-11-28 | MA     | 12
   (1 row)

   The value in month remains unchanged.

2. Refresh the default value in month from column date:

   => UPDATE t SET month=DEFAULT;
   OUTPUT
   -------
     1
   (1 row)

   => COMMIT;
   COMMIT

   => SELECT * FROM t;
   date  | state | month
   -------+-------+------
     2017-11-28 | MA     | 11
   (1 row)

Derive a default column value from user-defined scalar function

This example shows a user-defined scalar function that adds two integer values. The function is called add2ints and takes two arguments.

1. Develop and deploy the function, as described in Scalar Functions (UDSFs).

2. Create a sample table, t1, with two integer columns:

   => CREATE TABLE t1 ( x int, y int );
   CREATE TABLE

3. Insert some values into t1:
4. Use ALTER TABLE to add a column to t1, with the default column value derived from the UDSF add2ints:

```sql
insert into t1 values (1,2);
OUTPUT
--------
1
(1 row)
insert into t1 values (3,4);
OUTPUT
--------
1
(1 row)
```

```sql
alter table t1 add column z int default add2ints(x,y);
```

5. List the new column:

```sql
select z from t1;
z----
3
7
(2 rows)
```

**Table with a SET USING column that queries another table for its values**

1. Define tables t1 and t2. Column t2.b is defined to get its data from column t1.b, through the query in its SET USING clause:

```sql
=> CREATE TABLE t1 (a INT PRIMARY KEY ENABLED, b INT);
CREATE TABLE

=> CREATE TABLE t2 (a INT, alpha VARCHAR(10),
  b INT SET USING (SELECT t1.b FROM t1 WHERE t1.a=t2.a))
  ORDER BY a SEGMENTED BY HASH(a) ALL NODES;
CREATE TABLE
```

**Important:** The definition for table t2 includes SEGMENTED BY and ORDER BY clauses that exclude SET USING column b. If these clauses are omitted, Vertica creates an auto-projection for this table that specifies column b in its SEGMENTED BY and ORDER BY clauses. Inclusion of a SET USING column in any projection's segmentation or sort order prevents function REFRESH_COLUMNS from populating this column. Instead, it returns with an error.

For details on this and other restrictions, see REFRESH_COLUMNS.
2. Populate the tables with data:

```sql
=> INSERT INTO t1 VALUES(1,11);
=> INSERT INTO t1 VALUES(2,22);
=> INSERT INTO t1 VALUES(3,33);
=> INSERT INTO t1 VALUES(4,44);
=> INSERT INTO t2 VALUES(1,'aa');
=> INSERT INTO t2 VALUES(2,'bb');
=> COMMIT;
COMMIT
```

3. View the data in table t2: Column in SET USING column b is empty, pending invocation of Vertica function REFRESH_COLUMNS:

```sql
=> SELECT * FROM t2;
a | alpha | b
-+-------+-
1 | aa    |
2 | bb    |
(2 rows)
```

4. Refresh the column data in table t2 by calling function REFRESH_COLUMNS:

```sql
=> SELECT REFRESH_COLUMNS ('t2','b', 'REBUILD');
  REFRESH_COLUMNS
---------------------
  refresh_columns completed
  (1 row)
```

In this example, REFRESH_COLUMNS is called with the optional argument REBUILD. This argument specifies to replace all data in SET USING column b. It is generally good practice to call REFRESH_COLUMNS with REBUILD on any new SET USING column. For details, see REFRESH_COLUMNS.

5. View data in refreshed column b, whose data is obtained from table t1 as specified in the column's SET USING query:

```sql
=> SELECT * FROM t2 ORDER BY a;
a | alpha | b
-+-------+-
1 | aa    | 11
2 | bb    | 22
(2 rows)
```

### Altering Table Definitions

Using ALTER TABLE syntax, you can respond to your evolving database schema requirements. The ability to change the definition of existing database objects facilitates ongoing
maintenance. Furthermore, most of these options are both fast and efficient for large tables, because they consume fewer resources and less storage than having to stage data in a temporary table.

`ALTER TABLE` lets you perform the following table-level changes:

- Add and drop table columns.
- Rename a table.
- Add and drop constraints.
- Alter key constraint enforcement.
- Move a table to a new schema.
- Change a table owner.
- Change and reorganize table partitions.

`ALTER TABLE` has an `ALTER COLUMN` clause that lets you modify column definitions—for example, change their name or data type. For column-level changes, see Managing Table Columns.

### Exclusive `ALTER TABLE` Clauses

The following `ALTER TABLE` clauses are exclusive: you cannot combine them with another `ALTER TABLE` clause:

- ADD COLUMN
- RENAME COLUMN
- SET SCHEMA
- PARTITION BY
- REORGANIZE
- REMOVE PARTITIONING
- RENAME [TO]
- OWNER TO
Note: You can use the ADD constraints and DROP constraints clauses together.

Using Consecutive ALTER TABLE Commands

With the exception of performing a table rename, complete ALTER TABLE statements consecutively. For example, to add multiple columns to a table, issue consecutive ALTER TABLE ADD COLUMN statements.

External Table Restrictions

Not all ALTER TABLE options pertain to external tables. For instance, you cannot add a column to an external table, but you can rename the table:

```sql
=> ALTER TABLE mytable RENAME TO mytable2;
ALaR TABLE
```

Restoring to an Earlier Epoch

If you restore the database to an epoch that precedes changes to the table definition, the restore operation reverts the table to the earlier definition. For example, if you change a column's data type from CHAR(8) to CHAR(16) in epoch 10, and then restore the database from epoch 5, the column reverts to CHAR(8) data type.

Adding Table Columns

You add a column to a persistent table with the ALTER TABLE clause ADD COLUMN:

```sql
ALTER TABLE
... ADD COLUMN column-name datatype
[column-constraint]
[ENCODING encoding-type]
[RESTRICT | CASCADE]
[PROJECTIONS (projection-name [,....])]
```

Note: Before can add columns to a table, verify that all superprojections that are anchored to it are up to date.
Table Locking

When you use ADD COLUMN to alter a table, Vertica takes an O lock on the table until the operation completes. The lock prevents DELETE, UPDATE, INSERT, and COPY statements from accessing the table. The lock also blocks SELECT statements issued at SERIALIZABLE isolation level, until the operation completes.

If you use CASCADE, Vertica also takes O locks on all anchor tables of any pre-join projections associated with the target table. Consequently, SELECT statements issued on those tables at SERIALIZABLE isolation level are blocked until the operation completes.

Adding a column to a table does not affect K-safety of the physical schema design.

You can add columns when nodes are down.

Adding New Columns to Projections

When you add a column to a table, Vertica automatically adds the column to superprojections of that table. The ADD...COLUMN clause can also include the PROJECTIONS option, which specifies to add the column to one or more non-super projections.

For example, the store_orders table has two projections—superprojection store_orders_super, and user-created projection store_orders_p. The following ALTER TABLE...ADD COLUMN statement adds column expected_ship_date to the store_orders table. Because the statement omits the PROJECTIONS option, Vertica adds the column only to the table's superprojection:

```
=> ALTER TABLE public.store_orders ADD COLUMN expected_ship_date date;
=> SELECT projection_column_name, projection_name FROM projection_columns WHERE table_name ILIKE 'store_orders'
    ORDER BY projection_name , projection_column_name;

+------------+----------------+
| projection_column_name | projection_name |
|-----------------------+----------------|
| order_date            | store_orders_p_b0 |
| order_no              | store_orders_p_b0 |
| ship_date             | store_orders_p_b0 |
| order_date            | store_orders_p_b1 |
| order_no              | store_orders_p_b1 |
| ship_date             | store_orders_p_b1 |
| expected_ship_date    | store_orders_super |
| order_date            | store_orders_super |
| order_no              | store_orders_super |
| ship_date             | store_orders_super |
| shipper               | store_orders_super |
(11 rows)
```
The following ALTER TABLE...ADD COLUMN statement includes the PROJECTIONS option. This specifies to include projection store_orders_p in the add operation. Vertica adds the new column to this projection and the table's superprojection:

```
=> ALTER TABLE public.store_orders ADD COLUMN delivery_date date PROJECTIONS (store_orders_p);
=> SELECT projection_column_name, projection_name FROM projection_columns WHERE table_name ILIKE 'store_orders'
  ORDER BY projection_name, projection_column_name;
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>delivery_date</td>
<td>store_orders_p_b0</td>
</tr>
<tr>
<td>order_date</td>
<td>store_orders_p_b0</td>
</tr>
<tr>
<td>order_no</td>
<td>store_orders_p_b0</td>
</tr>
<tr>
<td>ship_date</td>
<td>store_orders_p_b0</td>
</tr>
<tr>
<td>delivery_date</td>
<td>store_orders_p_b1</td>
</tr>
<tr>
<td>order_date</td>
<td>store_orders_p_b1</td>
</tr>
<tr>
<td>order_no</td>
<td>store_orders_p_b1</td>
</tr>
<tr>
<td>ship_date</td>
<td>store_orders_p_b1</td>
</tr>
<tr>
<td>delivery_date</td>
<td>store_orders_super</td>
</tr>
<tr>
<td>expected_ship_date</td>
<td>store_orders_super</td>
</tr>
<tr>
<td>order_date</td>
<td>store_orders_super</td>
</tr>
<tr>
<td>order_no</td>
<td>store_orders_super</td>
</tr>
<tr>
<td>ship_date</td>
<td>store_orders_super</td>
</tr>
<tr>
<td>shipper</td>
<td>store_orders_super</td>
</tr>
</tbody>
</table>

(14 rows)

Updating Associated Table Views

Adding new columns to a table that has an associated view does not update the view's result set, even if the view uses a wildcard (*) to represent all table columns. To incorporate new columns, you must recreate the view.

Dropping Table Columns

When an ALTER TABLE statement includes a DROP COLUMN clause to drop a column, Vertica drops the specified column from the table and the ROS containers that correspond to the dropped column.

The syntax looks like this:

```
ALTER TABLE [schema.]table-name DROP [ COLUMN ] column-name [ CASCADE | RESTRICT ]
```

After a DROP COLUMN operation completes, data backed up from the current epoch onward recovers without the column. Data recovered from a backup that precedes the current epoch re-add the table column. Because drop operations physically purge object storage and catalog definitions (table history) from the table, AT EPOCH (historical) queries return nothing for the dropped column.

The altered table retains its object ID.
**Note:** Drop column operations can be fast because these catalog-level changes do not require data reorganization, so Vertica can quickly reclaim disk storage.

**Restrictions**

- You cannot drop or alter a primary key column or a column that participates in the table partitioning clause.

- You cannot drop the first column of any projection sort order, or columns that participate in a projection segmentation expression.

- All nodes must be up.

- You cannot drop a column associated with an access policy. Attempts to do so produce the following error:
  
  ```
  ERROR 6482: Failed to parse Access Policies for table "t1"
  ```

**Using CASCADE to Force a Drop**

If the table column to drop has dependencies, you must qualify the `DROP COLUMN` clause with the `CASCADE` option. For example, the target column might be specified in a projection sort order. In this and other cases, `DROP COLUMN . . CASCADE` handles the dependency by reorganizing catalog definitions or dropping a projection. In all cases, `CASCADE` performs the minimal reorganization required to drop the column.

Use `CASCADE` to drop a column with the following dependencies:

<table>
<thead>
<tr>
<th>Dropped column dependency</th>
<th>CASCADE behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any constraint</td>
<td>Vertica drops the column when a FOREIGN KEY constraint depends on a UNIQUE or PRIMARY KEY constraint on the referenced columns.</td>
</tr>
<tr>
<td>Specified in projection sort order</td>
<td>Vertica truncates projection sort order up to and including the projection that is dropped without impact on physical storage for other columns and then drops the specified column. For example if a projection's columns are in sort order (a,b,c), dropping column b causes the projection's sort order to be just (a), omitting column (c).</td>
</tr>
<tr>
<td>Specified in one of the</td>
<td>In both cases, the column to drop is integral to the</td>
</tr>
</tbody>
</table>
Dropped column dependency | CASCADE behavior
--- | ---
following:  
- Pre-join projection  
- Projection segmentation expression | projection definition. If possible, Vertica drops the projections as long as doing so does not compromise K-safety; otherwise, the transaction rolls back.  
For example, a table has multiple projections, where one projection's segmentation clause specifies the target column. Vertica tries to drop this projection, unless doing so violates K-safety. In this case, the transaction rolls back.

Referenced as default value of another column | See Dropping a Column Referenced as Default, below.

Dropping a Column Referenced as Default

You might want to drop a table column that is referenced by another column as its default value. For example, the following table is defined with two columns, a and b:, where b gets its default value from column a:

```sql
=> CREATE TABLE x (a int) UNSEGMENTED ALL NODES;
CREATE TABLE
=> ALTER TABLE x ADD COLUMN b int DEFAULT a;
ALTER TABLE
```

In this case, dropping column a requires the following procedure:

1. Remove the default dependency through `ALTER COLUMN . . DROP DEFAULT`:

   ```sql
   => ALTER TABLE x ALTER COLUMN b DROP DEFAULT;
   ```

2. Create a replacement superprojection for the target table if one or both of the following conditions is true:

   - The target column is the table's first sort order column. If the table has no explicit sort order, the default table sort order specifies the first table column as the first sort order column. In this case, the new superprojection must specify a sort order that excludes the target column.
If the table is segmented, the target column is specified in the segmentation expression. In this case, the new superprojection must specify a segmentation expression that excludes the target column.

Given the previous example, table x has a default sort order of (a,b). Because column a is the table's first sort order column, you must create a replacement superprojection that is sorted on column b:

```sql
=> CREATE PROJECTION x_p1 as select * from x order by b unsegmented all nodes;
```

3. Run START_REFRESH:

```sql
=> SELECT START_REFRESH();
START_REFRESH
Starting refresh background process.
(1 row)
```

4. Run MAKE_AHM_NOW:

```sql
=> SELECT MAKE_AHM_NOW();
MAKE_AHM_NOW
AHM set (New AHM Epoch: 1231)
(1 row)
```

5. Drop the column:

```sql
=> ALTER TABLE x DROP COLUMN a CASCADE;
```

Vertica implements the CASCADE directive as follows:

- Drops the original superprojection for table x (x_super).
- Updates the replacement superprojection x_p1 by dropping column a.

**Examples**

The following series of commands successfully drops a BYTEA data type column:

```sql
=> CREATE TABLE t (x BYTEA(65000), y BYTEA, z BYTEA(1));
CREATE TABLE
=> ALTER TABLE t DROP COLUMN y;
ALTER TABLE
=> SELECT y FROM t;
ERROR 2624: Column "y" does not exist
```
The following series of commands tries to drop a FLOAT(8) column and fails because there are not enough projections to maintain K-safety.

ALTER TABLE t DROP COLUMN x RESTRICT;
ALTER TABLE
=> SELECT x FROM t;
ERROR 2624: Column "x" does not exist
=> SELECT * FROM t;
  z
---
(0 rows)
=> DROP TABLE t CASCADE;
DROP TABLE

ALTER TABLE
=> SELECT y FROM t;
ERROR 2624: Column "y" does not exist
=> ALTER TABLE t DROP COLUMN y RESTRICT;
=> ALTER TABLE
=> SELECT y FROM t;
ERROR 2624: Column "y" does not exist
=> ALTER TABLE t DROP x CASCADE;
=> ROLLBACK 2409: Cannot drop any more columns in t
=> DROP TABLE t CASCADE;

### Altering Constraint Enforcement

To alter how Vertica enforces constraints, use the **ALTER TABLE** clause **ALTER CONSTRAINT**. You must qualify this clause with the keyword **ENABLED** or **DISABLED**:

- **ENABLED** automatically enforces a primary key, unique key or check constraint.
- **DISABLED** prevents automatic enforcement of a primary key, unique key or check constraint.

For more information on automatic constraint enforcement, see [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).

If you disable automatic enforcement of primary or unique key constraints, you can instead run **ANALYZE_CONSTRAINTS** to verify that columns have unique values after running a DML command or bulk loading. In the case of check constraints, you can use **ANALYZE_CONSTRAINTS** to validate check constraint conditions.

See [About Constraints](#) for general information about constraints.
Renaming Tables

The `ALTER TABLE RENAME TO` statement lets you rename one or more tables. Renaming tables does not change the table OID.

You rename multiple tables by supplying two comma-delimited lists. Vertica maps the names according to their order in the two lists. Only the first list can qualify table names with a schema. For example:

```
=> ALTER TABLE S1.T1, S1.T2 RENAME TO U1, U2;
```

The `RENAME TO` parameter is applied atomically: all tables are renamed, or none of them. For example, if the number of tables to rename does not match the number of new names, none of the tables is renamed.

**Note:** Renaming a table referenced by a view causes the view to fail, unless you create another table with the previous name to replace the renamed table.

Using Rename to Swap Tables Within a Schema

You can use `ALTER TABLE RENAME TO` to swap tables within the same schema, without actually moving data. You cannot swap tables across schemas.

To swap tables within a schema (example statement is split to explain steps):

1. Enter the names of the tables to swap, followed by a new temporary table placeholder (temps):

   ```
   => ALTER TABLE T1, T2, temps
   ```

2. Use the `RENAME TO` clause to swap the tables: `T1` to `temps`, `T2` to `T1`, and `temps` to `T2`:

   ```
   RENAME TO temps, T1, T2;
   ```

Moving Tables to Another Schema

The `ALTER TABLE` clause `SET SCHEMA` moves a table from one schema to another. Vertica automatically moves all projections that are anchored to the source table to the destination
schema. It also moves all IDENTITY and AUTO_INCREMENT columns to the destination schema.

Moving a table across schemas requires that you have USAGE privileges on the current schema and CREATE privileges on destination schema. You can move only one table between schemas at a time. You cannot move temporary tables across schemas.

Name Conflicts

If a table of the same name or any of the projections that you want to move already exist in the new schema, the statement rolls back and does not move either the table or any projections. To work around name conflicts:

1. Rename any conflicting table or projections that you want to move.
2. Run the ALTER TABLE SET SCHEMA statement again.

Note: Vertica lets you move system tables to system schemas. Moving system tables could be necessary to support designs created through the Database Designer.

Example

The following example moves table T1 from schema S1 to schema S2. All projections that are anchored on table T1 automatically move to schema S2:

```sql
=> ALTER TABLE S1.T1 SET SCHEMA S2;
```

Changing Table Ownership

As a superuser or table owner, you can reassign table ownership with ALTER TABLE, as follows:

```
ALTER TABLE [schema.]table-name OWNER TO owner-name
```

Changing table ownership is useful when moving a table from one schema to another. Ownership reassignment is also useful when a table owner leaves the company or changes job responsibilities. Because you can change the table owner, the tables won't have to be completely rewritten, you can avoid loss in productivity.

Changing table ownership automatically causes the following changes:
Grants on the table that were made by the original owner are dropped and all existing privileges on the table are revoked from the previous owner. Changes in table ownership has no effect on schema privileges.

Ownership of dependent IDENTITY/AUTO_INCREMENT sequences are transferred with the table. However, ownership does not change for named sequences created with CREATE SEQUENCE. To transfer ownership of these sequences, use ALTER SEQUENCE.

New table ownership is propagated to its projections.

Example

In this example, user Bob connects to the database, looks up the tables, and transfers ownership of table t33 from himself to user Alice.

```sql
=> \c - Bob
You are now connected as user "Bob".
=> \d
  Schema     | Name     | Kind | Owner | Comment
-------------|----------|------|-------|------------------------
  public     | applog   | table | dbadmin |
  public     | t33      | table | Bob    |
              (2 rows)  
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
```

When Bob looks up database tables again, he no longer sees table t33:

```sql
=> \d
  List of tables
  List of tables
  Schema | Name | Kind | Owner | Comment
----------|------|------|-------|------------------------
  public  | applog | table | dbadmin |
              (1 row)  
```

When user Alice connects to the database and looks up tables, she sees she is the owner of table t33.

```sql
=> \c - Alice
You are now connected as user "Alice".
=> \d
  List of tables
  Schema | Name | Kind | Owner | Comment
----------|------|------|-------|------------------------
  public  | t33  | table | Alice |
              (2 rows)  
```

Alice or a superuser can transfer table ownership back to Bob. In the following case a superuser performs the transfer.
=> \c - dbadmin
You are now connected as user "dbadmin".
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> \d

| List of tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>applog</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>comments</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>t33</td>
<td>table</td>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>t1</td>
<td>table</td>
<td>User1</td>
<td></td>
</tr>
</tbody>
</table>
(4 rows)

You can also query system table V_CATALOG.TABLES to view table and owner information. Note that a change in ownership does not change the table ID.

In the below series of commands, the superuser changes table ownership back to Alice and queries the TABLES system table.

=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;

<table>
<thead>
<tr>
<th>table_schema_id</th>
<th>table_schema</th>
<th>table_id</th>
<th>table_name</th>
<th>owner_id</th>
<th>owner_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>4503596273713634</td>
<td>applog</td>
<td>45035996273704962</td>
<td>dbadmin</td>
</tr>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>4503596273724496</td>
<td>comments</td>
<td>45035996273704962</td>
<td>dbadmin</td>
</tr>
<tr>
<td>4503596273730528</td>
<td>s1</td>
<td>45035996273730548</td>
<td>t33</td>
<td>45035996273730516</td>
<td>User1</td>
</tr>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>45035996273795846</td>
<td>t33</td>
<td>45035996273724576</td>
<td>Alice</td>
</tr>
</tbody>
</table>
(5 rows)

Now the superuser changes table ownership back to Bob and queries the TABLES table again. Nothing changes but the owner_name row, from Alice to Bob.

=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM tables;

<table>
<thead>
<tr>
<th>table_schema_id</th>
<th>table_schema</th>
<th>table_id</th>
<th>table_name</th>
<th>owner_id</th>
<th>owner_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>45035996273713634</td>
<td>applog</td>
<td>45035996273704962</td>
<td>dbadmin</td>
</tr>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>45035996273724496</td>
<td>comments</td>
<td>45035996273704962</td>
<td>dbadmin</td>
</tr>
<tr>
<td>4503596273730528</td>
<td>s1</td>
<td>45035996273730548</td>
<td>t1</td>
<td>45035996273730516</td>
<td>User1</td>
</tr>
<tr>
<td>4503596273704968</td>
<td>public</td>
<td>45035996273795846</td>
<td>t33</td>
<td>45035996273714428</td>
<td>Bob</td>
</tr>
</tbody>
</table>
(5 rows)
Sequence Types

Vertica supports the following sequence types:

- `AUTO_INCREMENT/IDENTITY` column sequences: Column constraints `AUTO_INCREMENT` and `IDENTITY` are synonyms. They specify to increment or decrement a column’s value as new rows are added. This sequence type is table-dependent and does not persist independently. A table can contain only one `AUTO_INCREMENT` or `IDENTITY` column. For detailed information, see AUTO_INCREMENT and IDENTITY Sequences.

- Named sequences: A named sequence is a database object that generates unique numbers in sequential ascending or descending order. Named sequences are defined independently through `CREATE SEQUENCE` statements, and are managed independently of the tables that reference them. A table can set the default values of one or more columns to named sequences. For details, see Named Sequences.

Named Sequences Versus AUTO_INCREMENT/IDENTITY Sequences

The following table lists the differences between the two sequence types:

<table>
<thead>
<tr>
<th>Supported Behavior</th>
<th>Named Sequence</th>
<th>AUTO_INCREMENT/IDENTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default cache value 250K</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Set initial cache</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Define start value</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Specify increment unit</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Exists as an independent object</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Exists only as part of table</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Create as column constraint</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Requires name</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Use in expressions</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Named Sequences

Named sequences are sequences that are defined by CREATE SEQUENCE. While you can set the value of a table column to a named sequence, a named sequence, unlike AUTO_INCREMENT and IDENTITY sequences, exists independently of the table.

Named sequences are used most often when an application requires a unique identifier in a table or an expression. After a named sequence returns a value, it never returns the same value again in the same session.

Creating and Using Named Sequences

You create a named sequence with CREATE SEQUENCE. The statement requires only a sequence name; all other parameters are optional. To create a sequence, a user must have CREATE privileges on a schema that contains the sequence.

The following example creates an ascending named sequence, my_seq, starting at the value 100:

```
```
CREATE SEQUENCE my_seq START 100;

Incrementing and Decrementing a Sequence

When you create a named sequence object, you can also specify its increment or decrement value by setting its INCREMENT parameter. If you omit this parameter, as in the previous example, the default is set to 1.

You increment or decrement a sequence by calling the function NEXTVAL on it—either directly on the sequence itself, or indirectly by adding new rows to a table that references the sequence. When called for the first time on a new sequence, NEXTVAL initializes the sequence to its start value. Vertica also creates a cache for the sequence. Subsequent NEXTVAL calls on the sequence increment its value.

The following call to NEXTVAL initializes the new my_seq sequence to 100:

```sql
=> SELECT NEXTVAL('my_seq');
nextval
--------
   100
(1 row)
```

Getting a Sequence's Current Value

You can obtain the current value of a sequence by calling CURRVAL on it. For example:

```sql
=> SELECT CURRVAL('my_seq');
CURRVAL
--------
    100
(1 row)
```

**Note:** CURRVAL returns an error if you call it on a new sequence that has not yet been initialized by NEXTVAL, or an existing sequence that has not yet been accessed in a new session. For example:

```sql
=> CREATE SEQUENCE seq2;
CREATE SEQUENCE
=> SELECT currval('seq2');
ERROR 4700: Sequence seq2 has not been accessed in the session
```

Using Sequences in Tables

A table can set the default values of one or more columns to named sequences. In the following example, column id gets its default values from named sequence my_seq:
CREATE TABLE customer(id INTEGER DEFAULT my_seq.NEXTVAL,
  lname VARCHAR(25),
  fname VARCHAR(25),
  membership_card INTEGER)
);

For each row that you insert into table customer, the sequence invokes the NEXTVAL function to set the value of the id column. For example:

=> INSERT INTO customer VALUES (default, 'Carr', 'Mary', 87432);
=> INSERT INTO customer VALUES (default, 'Diem', 'Nga', 87433);
=> COMMIT;

For each row, the insert operation invokes NEXTVAL on the sequence my_seq, which increments the sequence to 101 and 102, and sets the id column to those values:

=> SELECT * FROM customer;

<table>
<thead>
<tr>
<th>id</th>
<th>lname</th>
<th>fname</th>
<th>membership_card</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Carr</td>
<td>Mary</td>
<td>87432</td>
</tr>
<tr>
<td>102</td>
<td>Diem</td>
<td>Nga</td>
<td>87433</td>
</tr>
</tbody>
</table>

(1 row)

Distributing Named Sequences

When you create a named sequence, its CACHE parameter determines the number of sequence values each node maintains during a session. The default cache value is 250K, so each node reserves 250,000 values per session for each sequence. The default cache size provides an efficient means for large insert or copy operations.

If sequence caching is set to a lower number, nodes are liable to request a new set of cache values more frequently. While it supplies new cache, Vertica must lock the catalog. Until Vertica releases the lock, other database activities such as table inserts are blocked, which can adversely affect overall performance.

When a new session starts, node caches are initially empty. By default, the initiator node requests and reserves cache for all nodes in a cluster. You can change this default so each node requests its own cache, by setting configuration parameter ClusterSequenceCacheMode to 0.

For information on how Vertica requests and distributes cache among all nodes in a cluster, refer to Sequence Caching.
Effects of Distributed Sessions

Vertica distributes a session across all nodes. The first time a cluster node calls the function `NEXTVAL` on a sequence to increment (or decrement) its value, the node requests its own cache of sequence values. The node then maintains that cache for the current session. As other nodes call `NEXTVAL`, they too create and maintain their own cache of sequence values.

During a session, nodes call `NEXTVAL` independently and at different frequencies. Each node uses its own cache to populate the sequence. All sequence values are guaranteed to be unique, but can be out of order with a `NEXTVAL` statement executed on another node. As a result, sequence values are often non-contiguous.

In all cases, increments a sequence only once per row. Thus, if the same sequence is referenced by multiple columns, `NEXTVAL` sets all columns in that row to the same value. This applies to rows of joined tables.

Calculating Named Sequences

Vertica calculates the current value of a sequence as follows:

- At the end of every statement, the state of all sequences used in the session is returned to the initiator node.

- The initiator node calculates the maximum `CURRVAL` of each sequence across all states on all nodes.

- This maximum value is used as `CURRVAL` in subsequent statements until another `NEXTVAL` is invoked.

Losing Sequence Values

Sequence values in cache can be lost in the following situations:

- If a statement fails after `NEXTVAL` is called (thereby consuming a sequence value from the cache), the value is lost.

- If a disconnect occurs (for example, dropped session), any remaining values in cache that have not been returned through `NEXTVAL` are lost.

- When the initiator node distributes a new block of cache to each node where one or more nodes has not used up its current cache allotment. For information on this scenario, refer to Sequence Caching.
You can recover lost sequence values by using `ALTERT SEQUENCE...RESTART`, which resets the sequence to the specified value in the next session.

Caution: Using `ALTER SEQUENCE` to set a sequence start value below its current value can result in duplicate keys.

Altering Sequences

`ALTER SEQUENCE` can change a named sequence in two ways:

- Reset parameters that control sequence behavior—for example, its start value, or range of minimum and maximum values. These changes take effect only when you start a new database session.

- Reset sequence name, schema, or ownership. These changes take effect immediately.

Note: The same `ALTER SEQUENCE` statement cannot make both types of changes.

Changing Sequence Behavior

`ALTER SEQUENCE` can change one or more sequence attributes through the following parameters:

<table>
<thead>
<tr>
<th>These parameters...</th>
<th>Control...</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENT</td>
<td>How much to increment or decrement the sequence on each call to <code>NEXTVAL</code>.</td>
</tr>
<tr>
<td>MINVALUE/MAXVALUE</td>
<td>The range of valid integers.</td>
</tr>
<tr>
<td>RESTART</td>
<td>The sequence value on its next call to <code>NEXTVAL</code>.</td>
</tr>
<tr>
<td>CACHE/NOCACHE</td>
<td>How many sequence numbers are pre-allocated and stored in memory for faster access.</td>
</tr>
<tr>
<td>CYCLE/NOCYCLE</td>
<td>Whether the sequence wraps when its minimum or maximum values are reached.</td>
</tr>
</tbody>
</table>

These changes take effect only when you start a new database session. For example, if you create a named sequence `my_sequence` that starts at 10 and increments by 1 (the default), each sequence call to `NEXTVAL` increments its value by 1:
=> CREATE SEQUENCE my_sequence START 10;
=> SELECT NEXTVAL('my_sequence');
   nextval
    -------
   10  
   (1 row)
=> SELECT NEXTVAL('my_sequence');
   nextval
    -------
   11  
   (1 row)

The following ALTER SEQUENCE statement specifies to restart the sequence at 50:

=> ALTER SEQUENCE my_sequence RESTART WITH 50;

However, this change has no effect in the current session. The next call to NEXTVAL increments the sequence to 12:

=> SELECT NEXTVAL('my_sequence');
   NEXTVAL
    -------
   12  
   (1 row)

The sequence restarts at 50 only after you start a new database session:

=> \q
$ vsql
Welcome to vsql, the Vertica Analytic Database interactive terminal.

=> SELECT NEXTVAL('my_sequence');
   NEXTVAL
    -------
   50  
   (1 row)

Changing Sequence Name, Schema, and Ownership

You can use ALTER SEQUENCE to make the following changes to a named sequence:

- Rename it.
- Move it to another schema.
- Reassign ownership.

Each of these changes requires separate ALTER SEQUENCE statements. These changes take effect immediately.

For example, the following statement renames a sequence from my_seq to serial:
This statement moves sequence `s1.serial` to schema `s2`:

```sql
ALTER SEQUENCE s1.my_seq RENAME TO s1.serial;
```

The following statement reassigns ownership of `s2.serial` to another user:

```sql
ALTER SEQUENCE s2.serial OWNER TO bertie;
```

**Note:** Only a superuser or the sequence owner can change its ownership. Reassignment does not transfer grants from the original owner to the new owner. Grants made by the original owner are dropped.

### Dropping Sequences

Use `DROP SEQUENCE` to remove a named sequence. For example:

```sql
DROP SEQUENCE my_sequence;
```

You cannot drop a sequence if one of the following conditions is true:

- Other objects depend on the sequence. `DROP SEQUENCE` does not support cascade operations.
- A column's `DEFAULT` expression references the sequence. Before dropping the sequence, you must remove all column references to it.

### AUTO_INCREMENT and IDENTITY Sequences

Column constraints `AUTO_INCREMENT` and `IDENTITY` are synonyms that associate a column with a sequence. This sequence automatically increments the column value as new rows are added.

You define an `AUTO_INCREMENT/IDENTITY` column in a table as follows:

```sql
CREATE TABLE table-name ...
    (column-name (AUTO_INCREMENT | IDENTITY) [(args)], ...)
```

where `args` is 1 to 3 optional arguments that let you control sequence behavior (see Arguments below).

`AUTO_INCREMENT/IDENTITY` sequences are owned by the table in which they are defined, and do not exist outside that table. Unlike named sequences, you cannot manage an `AUTO_INCREMENT/IDENTITY` sequence using `ALTER SEQUENCE`.

**Arguments**

- **`increment` (optional)**: Specifies the increment value, starting from `1`.
- **`min_value` (optional)**: Specifies the minimum value.
- **`max_value` (optional)**: Specifies the maximum value.
- **`cycle` (optional)**: Specifies whether the sequence wraps around after reaching `max_value`.

**Examples**

- `increment 5`: Increments by 5.
- `min_value 10`: Sets the minimum value to 10.
- `max_value 100`: Sets the maximum value to 100.
- `cycle 0`: Disables wrapping.
- `cycle 1`: Enables wrapping.

Note that the `cycle` argument is not available when using `IDENTITY`.

**Dropping AUTO_INCREMENT/IDENTITY columns**

You cannot drop a column with `AUTO_INCREMENT` or `IDENTITY` if you have any references to it in your database. You can drop the column from the table and then drop the sequence separately if necessary.

**Removing AUTO_INCREMENT/IDENTITY columns**

To remove `AUTO_INCREMENT` or `IDENTITY` constraints from a column, use the `ALTER COLUMN` command:

```sql
ALTER TABLE table-name
    ALTER COLUMN column-name TYPE (column-name NOT NULL AUTO_INCREMENT);
```

This removes the `AUTO_INCREMENT` constraint from the column, allowing you to set a new constraint or remove the column from the table.
INCREMENT/IDENTITY sequence with ALTER SEQUENCE. For example, you cannot change the schema of an AUTO_INCREMENT/IDENTITY sequence independently of its table. If you move the table to another schema, the sequence automatically moves with it.

You can obtain the last value generated for an AUTO_INCREMENT/IDENTITY sequence by calling Vertica meta-function LAST_INSERT_ID.

Arguments

AUTO_INCREMENT/IDENTITY constraints can take between 0 and three arguments. These arguments let you specify the column's start value, how much it increments or decrements, and how many unique numbers each node caches per session.

You specify these arguments as follows:

<table>
<thead>
<tr>
<th># arguments</th>
<th>Description</th>
</tr>
</thead>
</table>
| None        | The following default settings apply:  

  - The starting value is 1.  
  - Values increment by at least 1.  
  - Each node caches 250,000 unique numbers per session for this sequence. |
| 1           | Specifies how many unique numbers each node can cache per session, as follows:  

  - >1 specifies how many unique numbers each node caches per session.  
  - 0 or 1 specifies to disable caching.  

  **IDENTITY**(cache)  
  Default: 250,000 |
| 2 or 3      | Set as follows:  

  \{AUTO_INCREMENT \| IDENTITY\} (start, increment[, cache])\)  

  - **start**: The first value that is set for this column.  
    Default: 1  
  - **increment**: A positive or negative integer that specifies the minimum amount to increment or decrement the column value from its value in
the previous row.

Default: 1

Important: Setting this argument to a value of \(X\) guarantees that column values always increment by at least \(X\). However, column values can sometimes increment by more than \(X\) unless you also set the cache value to 0 or 1 (no cache).

- **cache**: One of the following:
  - \(>1\) specifies how many unique numbers each node can cache per session for this sequence.
  - 0 or 1 specifies to disable caching.

Default: 250,000

For details, see **Sequence Caching**.

### Restrictions

The following restrictions apply to **AUTO_INCREMENT/IDENTITY** columns:

- A table can contain only one **AUTO_INCREMENT/IDENTITY** column.

  Note: A table with an **AUTO_INCREMENT/IDENTITY** column can also contain one or more columns that are set to **named sequences**.

- **AUTO_INCREMENT/IDENTITY** values are never rolled back, even if a transaction that tries to insert a value into a table is not committed.

- You cannot change the value of an **AUTO_INCREMENT/IDENTITY** column.

### Examples

The following example shows how to use the **IDENTITY** column-constraint to create a table with an ID column. The ID column has an initial value of 1. It is incremented by 1 every time a row is inserted.
1. Create table Premium_Customer:

```sql
CREATE TABLE Premium_Customer(
    ID INT IDENTITY(1,1),
    lname VARCHAR(25),
    fname VARCHAR(25),
    store_membership_card INT
); INSERT INTO Premium_Customer (lname, fname, store_membership_card)
VALUES ('Gupta', 'Saleem', 475987);
```

The IDENTITY column has a seed of 1, which specifies the value for the first row loaded into the table, and an increment of 1, which specifies the value that is added to the IDENTITY value of the previous row.

2. Confirm the row you added and see the ID value:

```sql
SELECT * FROM Premium_Customer;
```

```
<table>
<thead>
<tr>
<th>ID</th>
<th>lname</th>
<th>fname</th>
<th>store_membership_card</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gupta</td>
<td>Saleem</td>
<td>475987</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
```

3. Add another row:

```sql
INSERT INTO Premium_Customer (lname, fname, store_membership_card)
VALUES ('Lee', 'Chen', 598742);
```

4. Call the Vertica function LAST_INSERT_ID. The function returns value 2 because you previously inserted a new customer (Chen Lee), and this value is incremented each time a row is inserted:

```sql
SELECT LAST_INSERT_ID();
```

```
<table>
<thead>
<tr>
<th>last_insert_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

(1 row)
```

5. View all the ID values in the Premium_Customer table:

```sql
SELECT * FROM Premium_Customer;
```

```
<table>
<thead>
<tr>
<th>ID</th>
<th>lname</th>
<th>fname</th>
<th>store_membership_card</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gupta</td>
<td>Saleem</td>
<td>475987</td>
</tr>
<tr>
<td>2</td>
<td>Lee</td>
<td>Chen</td>
<td>598742</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)
```

The next three examples illustrate the three valid ways to use IDENTITY arguments. These examples are valid for the AUTO_INCREMENT argument also.
The first example uses a cache of 100, and the defaults for start value (1) and increment value (1):

```sql
=> CREATE TABLE t1(x IDENTITY(100), y INT);
```

The next example specifies the start and increment values as 1, and defaults to a cache value of 250,000:

```sql
=> CREATE TABLE t2(y IDENTITY(1,1), x INT);
```

The third example specifies start and increment values of 1, and a cache value of 100:

```sql
=> CREATE TABLE t3(z IDENTITY(1,1,100), zx INT);
```

## Sequence Caching

Caching is similar for all sequence types: named sequences, identity sequences, and auto-increment sequences. To allocate cache among the nodes in a cluster for a given sequences, Vertica uses the following process.

1. By default, when a session begins, the cluster initiator node requests cache for itself and other nodes in the cluster.

2. The initiator node distributes cache to other nodes when it distributes the execution plan.

3. Because the initiator node requests caching for all nodes, only the initiator locks the global catalog for the cache request.

This approach is optimal for handling large INSERT-SELECT and COPY operations. The following figure shows how the initiator request and distributes cache for a named sequence in a three-node cluster, where caching for that sequence is set to 250 K:
Nodes run out of cache at different times. While executing the same query, nodes individually request additional cache as needed.

For new queries in the same session, the initiator might have an empty cache if it used all of its cache to execute the previous query execution. In this case, the initiator requests cache for all nodes.

You can change how nodes obtain sequence caches by setting the configuration parameter `ClusterSequenceCacheMode` to 0 (disabled). When this parameter is set to 0, all nodes in the cluster request their own cache and catalog lock. However, for initial large INSERT-SELECT and COPY operations, when the cache is empty for all nodes, each node requests cache at the same time. These multiple requests result in multiple simultaneous locks on the global catalog, which can adversely affect performance. For this reason, `ClusterSequenceCacheMode` should remain set to its default value of 1 (enabled).

The following example compares how different settings of `ClusterSequenceCacheMode` affect how Vertica manages sequence caching. The example assumes a three-node cluster, 250 K caches for each node (the default), and sequence ID values that increment by 1.
<table>
<thead>
<tr>
<th>Workflow step</th>
<th>ClusterSequenceCacheMode = 1</th>
<th>ClusterSequenceCacheMode = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cache is empty for all nodes. Initiator node requests 250 K cache for each node.</td>
<td>Cache is empty for all nodes. Each node, including initiator, requests its own 250 K cache.</td>
</tr>
</tbody>
</table>
| 2             | Blocks of cache are distributed to each node as follows:  
- Node 1: 0–250 K  
- Node 2: 250 K + 1 to 500 K  
- Node 3: 500 K + 1 to 750 K  
Each node begins to use its cache as it processes sequence updates. | |
| 3             | Initiator node and node 3 run out of cache.  
Node 2 only uses 250 K +1 to 400 K, 100 K of cache remains from 400 K +1 to 500 K. | |
| 4             | Executing same statement:  
- As each node uses up its cache, it requests a new cache allocation.  
- If node 2 never uses its cache, the 100-K unused cache becomes a gap in sequence IDs.  
Executing a new statement in same session, if initiator node cache is empty:  
- It requests and distributes new cache blocks for all nodes.  
- Nodes receive a new cache before the old cache is used, creating a gap in ID sequencing. | Executing same or new statement:  
- As each node uses up its cache, it requests a new cache allocation.  
- If node 2 never uses its cache, the 100 K unused cache becomes a gap in sequence IDs. |
Merging Table Data

MERGE statements can perform update and insert operations on a target table based on the results of a join with a source data set. The join can match a source row with only one target row; otherwise, Vertica returns an error.

MERGE has the following syntax:

```
MERGE INTO target-table USING source-dataset ON join-condition
    [ matching-clause [ matching-clause ] ]
```

Merge operations have at least three components:

- The target table on which to perform update and insert operations. MERGE takes an X (exclusive) lock on the target table until the merge operation is complete.

- Join to another data set, one of the following: a table, view, or subquery result set.

- One or both matching clauses: WHEN MATCHED THEN UPDATE SET and WHEN NOT MATCHED THEN INSERT.

Basic MERGE Example

In this example, a merge operation involves two tables:

- `visits_daily` logs daily restaurant traffic, and is updated with each customer visit. Data in this table is refreshed every 24 hours.

- `visits_history` stores the history of customer visits to various restaurants, accumulated over an indefinite time span.

Each night, you merge the daily visit count from `visits_daily` into `visits_history`. The merge operation modifies the target table in two ways:

- Updates existing customer data.

- Inserts new rows of data for first-time customers.

One MERGE statement executes both operations as a single (upsert) transaction.
Source and Target Tables

The source and target tables visits_daily and visits_history are defined as follows:

```sql
CREATE TABLE public.visits_daily
(
    customer_id int,
    location_name varchar(20),
    visit_time time(0) DEFAULT (now())::timetz
);

CREATE TABLE public.visits_history
(
    customer_id int,
    location_name varchar(20),
    visit_count int
);
```

Table visits_history contains rows of three customers who between them visited two restaurants, Etoile and LaRosa:

```sql
=> SELECT * FROM visits_history ORDER BY customer_id, location_name;

<table>
<thead>
<tr>
<th>customer_id</th>
<th>location_name</th>
<th>visit_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Etoile</td>
<td>2</td>
</tr>
<tr>
<td>1002</td>
<td>La Rosa</td>
<td>4</td>
</tr>
<tr>
<td>1004</td>
<td>Etoile</td>
<td>1</td>
</tr>
</tbody>
</table>

(3 rows)
```

By close of business, table visits_daily contains three rows of restaurant visits:

```sql
=> SELECT * FROM visits_daily ORDER BY customer_id, location_name;

<table>
<thead>
<tr>
<th>customer_id</th>
<th>location_name</th>
<th>visit_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Etoile</td>
<td>18:19:29</td>
</tr>
<tr>
<td>1003</td>
<td>Lux Cafe</td>
<td>08:07:00</td>
</tr>
<tr>
<td>1004</td>
<td>La Rosa</td>
<td>11:49:20</td>
</tr>
</tbody>
</table>

(3 rows)
```

Table Data Merge

The following MERGE statement merges visits_daily data into visits_history:

- For matching customers, MERGE updates the occurrence count.
- For non-matching customers, MERGE inserts new rows.
MERGE INTO visits_history h USING visits_daily d
ON (h.customer_id=d.customer_id AND h.location_name=d.location_name)
WHEN MATCHED THEN UPDATE SET visit_count = h.visit_count + 1
WHEN NOT MATCHED THEN INSERT (customer_id, location_name, visit_count)
VALUES (d.customer_id, d.location_name, 1);

OUTPUT
-------
 3
(1 row)

MERGE returns the number of rows updated and inserted. In this case, the returned value specifies three updates and inserts:

- Customer 1001's third visit to Etoile
- New customer 1003's first visit to new restaurant Lux Cafe
- Customer 1004's first visit to La Rosa

If you now query table visits_history, the result set shows the merged (updated and inserted) data. Updated and new rows are highlighted:

```
=> SELECT * FROM visits_history ORDER BY customer_id, location_name
+---------+----------+----------+
| customer_id | location_name | visit_count |
+---------+----------+----------+
| 1001    | Etoile   | 3        |
| 1002    | La Rosa  | 4        |
| 1003    | Lux Cafe | 1        |
| 1004    | Etoile   | 1        |
| 1004    | La Rosa  | 1        |
+---------+----------+----------+
(5 rows)
```

**MERGE Source Options**

A MERGE operation joins the target table to one of the following data sources:

- Another table
- View
- Subquery result set

**Merging from Table and View Data**

You merge data from one table into another as follows:
MERGE INTO target-table USING (source-table | source-view) join-condition
  matching-clause[ matching-clause ]

If you specify a view, Vertica expands the view name to the query that it encapsulates, and uses the result set as the merge source data.

For example, the VMart table public.product_dimension contains current and discontinued products. You can move all discontinued products into a separate table public.product_dimension_discontinued, as follows:

```sql
=> CREATE TABLE public.product_dimension_discontinued (  
  product_key int,
  product_version int,
  sku_number char(32),
  category_description char(32),
  product_description varchar(128));
```

```sql
=> MERGE INTO product_dimension_discontinued tgt
  USING product_dimension src ON tgt.product_key = src.product_key
    AND tgt.product_version = src.product_version
  WHEN NOT MATCHED AND src.discontinued_flag='1' THEN INSERT VALUES
    (src.product_key,
    src.product_version,
    src.sku_number,
    src.category_description,
    src.product_description);
```

Output:
```
1186
(1 row)
```

Source table `product_dimension` uses two columns, `product_key` and `product_version`, to identify unique products. The MERGE statement joins the source and target tables on these columns in order to return single instances of non-matching rows. The WHEN NOT MATCHED clause includes a filter `(src.discontinued_flag='1')`, which reduces the result set to include only discontinued products. The remaining rows are inserted into target table `product_dimension_discontinued`.

**Merging from a Subquery Result Set**

You can merge into a table the result set that is returned by a subquery, as follows:

MERGE INTO target-table USING (subquery) sq-alias join-condition
  matching-clause[ matching-clause ]

For example, the VMart table `public.product_dimension` is defined as follows (DDL truncated):

```sql
CREATE TABLE public.product_dimension (  
  product_key int NOT NULL,
  product_version int NOT NULL,
  ...)
```
ALTER TABLE public.product_dimension ADD CONSTRAINT C_PRIMARY PRIMARY KEY (product_key, product_version) DISABLED;

Columns `product_key` and `product_version` comprise the table's primary key. You can modify this table so it contains a single column that concatenates the values of these two columns. This column can be used to uniquely identify each product, while also maintaining the original values from `product_key` and `product_version`.

You populate the new column with a MERGE statement that queries the other two columns:

```sql
ALTER TABLE public.product_dimension ADD COLUMN product_ID numeric(8,2);
MERGE INTO product_dimension tgt
USING (SELECT (product_key||'.0'||product_version)::numeric(8,2) AS pid, sku_number
FROM product_dimension) src
ON tgt.product_key||'.0'||product_version::numeric=src.pid
WHEN MATCHED THEN UPDATE SET product_ID = src.pid;
```

The following query verifies that the new column values correspond to the values in `product_key` and `product_version`:

```sql
SELECT product_ID, product_key, product_version, product_description
FROM product_dimension
WHERE category_description = 'Medical'
  AND product_description ILIKE '%diabetes%'
  AND discontinued_flag = 1 ORDER BY product_ID;
```

<table>
<thead>
<tr>
<th>product_ID</th>
<th>product_key</th>
<th>product_version</th>
<th>product_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5836.02</td>
<td>5836</td>
<td>2</td>
<td>Brand #17487 diabetes blood testing kit</td>
</tr>
<tr>
<td>14320.02</td>
<td>14320</td>
<td>2</td>
<td>Brand #43046 diabetes blood testing kit</td>
</tr>
<tr>
<td>18881.01</td>
<td>18881</td>
<td>1</td>
<td>Brand #56743 diabetes blood testing kit</td>
</tr>
</tbody>
</table>

(3 rows)

**MERGE Matching Clauses**

MERGE supports one instance of the following matching clauses:

- **WHEN MATCHED THEN UPDATE SET**
- **WHEN NOT MATCHED THEN INSERT**

Each matching clause can specify an additional filter, as described in Update and Insert Filters.
**WHEN MATCHED THEN UPDATE**

Updates all target table rows that are joined to the source table, typically with data from the source table:

\[
\text{WHEN MATCHED [ AND update-filter ] THEN UPDATE} \quad \text{SET } \{ \text{target-column} = \text{expression} \}[,...]
\]

Vertica can execute the join only on unique values in the source table's join column. If the source table's join column contains more than one matching value, the MERGE statement returns with a run-time error.

**WHEN NOT MATCHED THEN INSERT**

WHEN NOT MATCHED THEN INSERT inserts into the target table a new row for each source table row that is excluded from the join:

\[
\text{WHEN NOT MATCHED [ AND insert-filter ] THEN INSERT} \quad \left[ ( \text{column-list} ) \right] \text{VALUES ( values-list )}
\]

*column-list* is a comma-delimited list of one or more target columns in the target table, listed in any order. MERGE maps *column-list* columns to *values-list* values in the same order, and each column-value pair must be compatible. If you omit *column-list*, Vertica maps *values-list* values to columns according to column order in the table definition.

For example, given the following source and target table definitions:

```sql
CREATE TABLE t1 (a int, b int, c int);
CREATE TABLE t2 (x int, y int, z int);
```

The following WHEN NOT MATCHED clause implicitly sets the values of the target table columns a, b, and c in the newly inserted rows:

```sql
MERGE INTO t1 USING t2 ON t1.a=t2.x
    WHEN NOT MATCHED THEN INSERT VALUES (t2.x, t2.y, t2.z);
```

In contrast, the following WHEN NOT MATCHED clause excludes columns t1.b and t2.y from the merge operation. The WHEN NOT MATCHED clause explicitly pairs two sets of columns from the target and source tables: t1.a to t2.x, and t1.c to t2.z. Vertica sets excluded column t1.b to null:

```sql
MERGE INTO t1 USING t2 ON t1.a=t2.x
    WHEN NOT MATCHED THEN INSERT (a, c) VALUES (t2.x, t2.z);
```
Update and Insert Filters

Each `WHEN MATCHED` and `WHEN NOT MATCHED` clause in a MERGE statement can optionally specify an update filter and insert filter, respectively:

```
WHEN MATCHED AND update-filter THEN UPDATE ...
WHEN NOT MATCHED AND insert-filter THEN INSERT ...
```

Vertica also supports Oracle syntax for specifying update and insert filters:

```
WHEN MATCHED THEN UPDATE SET column-updates WHERE update-filter
WHEN NOT MATCHED THEN INSERT column-values WHERE insert-filter
```

Each filter can specify multiple conditions. Vertica handles the filters as follows:

- An update filter is applied to the set of matching rows in the target table that are returned by the MERGE join. For each row where the update filter evaluates to true, Vertica updates the specified columns.

- An insert filter is applied to the set of source table rows that are excluded from the MERGE join. For each row where the insert filter evaluates to true, Vertica adds a new row to the target table with the specified values.

For example, given the following data in tables `t11` and `t22`:

```
=> SELECT * from t11 ORDER BY pk;
  pk | col1 | col2 | SKIP_ME_FLAG
+----+------|------|-----------------
  1  |  2   |  3   | t
  2  |  3   |  4   | t
  3  |  4   |  5   | f
  4  |     |  6   | f
  5  |  6   |  7   | t
  6  |     |  8   | f
  7  |  8   |     | t
(7 rows)
```

```
=> SELECT * FROM t22 ORDER BY pk;
  pk | col1 | col2
+----+------|------
  1  |  2   |  4
  2  |  4   |  8
  3  |  6   |   
  4  |  8   | 16
(4 rows)
```

You can merge data from table `t11` into table `t22` with the following MERGE statement, which includes update and insert filters:
MERGE INTO t22 USING t11 ON ( t11.pk=t22.pk )
   WHEN MATCHED
       AND t11.SKIP_ME_FLAG=FALSE AND ( COALESCE (t22.col1<>t11.col1, (t22.col1 is null)<>(t11.col1 is null))
   )
       THEN UPDATE SET col1=t11.col1, col2=t11.col2
   WHEN NOT MATCHED
       AND t11.SKIP_ME_FLAG=FALSE
       THEN INSERT (pk, col1, col2) VALUES (t11.pk, t11.col1, t11.col2);
OUTPUT
--------
   3
(1 row)

=> SELECT * FROM t22 ORDER BY pk;
pk | col1 | col2
--------
   1 |   2 |  4
   2 |   4 |  8
   3 |   4 |  5
   4 |   5 |
   6 |   8 |
(5 rows)

Vertica uses the update and insert filters as follows:

- Evaluates all matching rows against the update filter conditions. Vertica updates each row where the following two conditions both evaluate to true:
  - Source column t11.SKIP_ME_FLAG is set to false.
  - The COALESCE function evaluates to true.

- Evaluates all non-matching rows in the source table against the insert filter. For each row where column t11.SKIP_ME_FLAG is set to false, Vertica inserts a new row in the target table.

**MERGE Optimization**

You can improve MERGE performance in several ways:

- **Design projections for optimal MERGE performance.**

- **Facilitate creation of optimized query plans.**

- Use source tables that are smaller than target tables.
Projections for MERGE Operations

The Vertica query optimizer automatically chooses the best projections to implement a merge operation. A good projection design strategy provides projections that help the query optimizer avoid extra sort and data transfer operations, and facilitate MERGE performance.

Tip: You can rely on Database Designer to generate projections that address merge requirements. You can then customize these projections as needed.

For example, the following MERGE statement fragment joins source and target tables tgt and src, respectively, on columns tgt.a and src.b:

```sql
=> MERGE INTO tgt USING src ON tgt.a = src.b ...
```

Vertica can use a local merge join if projections for tables tgt and src use one of the following projection designs, where inputs are presorted by projection ORDER BY clauses:

- Replicated projections are sorted on:
  - Column a for table tgt
  - Column b for table src

- Segmented projections are identically segmented on:
  - Column a for table tgt
  - Column b for table src
  - Corresponding segmented columns

Optimizing MERGE Query Plans

Vertica prepares an optimized query plan if the following conditions are all true:

- The MERGE statement contains both matching clauses `WHEN MATCHED THEN UPDATE SET` and `WHEN NOT MATCHED THEN INSERT`. If the MERGE statement contains only one matching clause, it uses a non-optimized query plan.

- The MERGE statement excludes update and insert filters.
- The target table join column has a unique or primary key constraint. This requirement does not apply to the source table join column.

- Both matching clauses specify all columns in the target table.

- Both matching clauses specify identical source values.

For details on evaluating an EXPLAIN-generated query plan, see MERGE Path.

The examples that follow use a simple schema to illustrate some of the conditions under which Vertica prepares or does not prepare an optimized query plan for MERGE:

```
CREATE TABLE target(a INT PRIMARY KEY, b INT, c INT) ORDER BY b,a;
CREATE TABLE source(a INT, b INT, c INT) ORDER BY b,a;
INSERT INTO target VALUES(1,2,3);
INSERT INTO target VALUES(2,4,7);
INSERT INTO source VALUES(3,4,5);
INSERT INTO source VALUES(4,6,9);
COMMIT;
```

**Optimized MERGE statement**

Vertica can prepare an optimized query plan for the following MERGE statement because:

- The target table's join column \( t.a \) has a primary key constraint.

- All columns in the target table \((a,b,c)\) are included in the UPDATE and INSERT clauses.

- The UPDATE and INSERT clauses specify identical source values: \(s.a\), \(s.b\), and \(s.c\).

```
MERGE INTO target t USING source s ON t.a = s.a
WHEN MATCHED THEN UPDATE SET a=s.a, b=s.b, c=s.c
WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a, s.b, s.c);
```

The output value of 2 indicates success and denotes the number of rows updated/inserted from the source into the target.

**Non-optimized MERGE statement**

In the next example, the MERGE statement runs without optimization because the source values in the UPDATE/INSERT clauses are not identical. Specifically, the UPDATE clause includes constants for columns \(s.a\) and \(s.c\) and the INSERT clause does not:

```
MERGE INTO target t USING source s ON t.a = s.a
WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c - 1
```
To make the previous MERGE statement eligible for optimization, rewrite the statement so that the source values in the UPDATE and INSERT clauses are identical:

```sql
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c -1
  WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a + 1, s.b, s.c - 1);
```

**MERGE Restrictions**

The following restrictions apply to updating and inserting table data with **MERGE**.

**Constraint Enforcement**

MERGE respects all enforced constraints in the target table. If the merge operation attempts to copy values that violate those constraints, MERGE returns with an error and rolls back the merge operation.

Caution: If you run MERGE multiple times using the same target and source table, each iteration is liable to introduce duplicate values into the target columns and return with an error.

**Columns Prohibited from Merge**

The following columns cannot be specified in a merge operation; attempts to do so return with an error:

- Identity/auto-increment columns, or columns whose default value is set to a named sequence.
- Vmap columns such as `__raw__` in flex tables.

**Removing Table Data**

Vertica provides several ways to remove data from a table:
Delete operation | Description
---|---
**DROP TABLE** | Permanently removes a table and its definition, optionally removes associated views and projections.
**DELETE** | Marks rows with delete vectors and stores them so data can be rolled back to a previous epoch. The data must be purged to reclaim disk space. See [Purging Deleted Data](#).
**TRUNCATE TABLE** | Removes all storage and history associated with a table. The table structure is preserved for future use. The results of this command cannot be rolled back.
**DROP PARTITIONS** | Removes one partition from a partitioned table. Each partition contains a related subset of data in the table. Partitioned data can be dropped efficiently, and provides query performance benefits. See [Partitioning Tables](#).

**Delete Operations Compared**

The following table summarizes differences between various delete operations.

<table>
<thead>
<tr>
<th>Operations and options</th>
<th>Performance</th>
<th>Auto commits</th>
<th>Saves history</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE FROM <em>table</em></td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE FROM <em>temp-table</em></td>
<td>High</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DELETE FROM <em>table where-clause</em></td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE FROM <em>temp-table where-clause</em></td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE FROM <em>temp-table where-clause</em> ON COMMIT PRESERVE ROWS</td>
<td>Normal</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE FROM <em>temp-table where-clause</em> ON COMMIT DELETE ROWS</td>
<td>High</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DROP <em>table</em></td>
<td>High</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TRUNCATE <em>table</em></td>
<td>High</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TRUNCATE <em>temp-table</em></td>
<td>High</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Choosing the Best Delete Operation

The following table can help you decide which delete operation is best:

<table>
<thead>
<tr>
<th>If you want to...</th>
<th>Use...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete both table data and definitions and start from scratch.</td>
<td><strong>DROP TABLE</strong></td>
</tr>
<tr>
<td>Quickly drop data while preserving table definitions, and reload data.</td>
<td><strong>TRUNCATE TABLE</strong></td>
</tr>
<tr>
<td>Regularly perform bulk delete operations.</td>
<td><strong>DROP_PARTITIONS</strong></td>
</tr>
<tr>
<td>Perform occasional small deletes or updates with the option to roll back or review history.</td>
<td><strong>DELETE</strong></td>
</tr>
</tbody>
</table>

See also: Best Practices for DELETE and UPDATE.
Best Practices for DELETE and UPDATE

Vertica is optimized for query-intensive workloads, so DELETE and UPDATE queries might not achieve the same level of performance as other queries. DELETE and UPDATE operations go to the WOS by default, but if the data is sufficiently large and would not fit in memory, Vertica automatically switches to using the ROS. See Using INSERT, UPDATE, and DELETE.

The topics that follow discuss best practices when using DELETE and UPDATE operations in Vertica.

DELETE and UPDATE Performance Considerations

To improve the performance of your DELETE and UPDATE queries, consider the following issues:

- **Query performance after large deletes**—A large number of (unpurged) deleted rows can negatively affect query performance.

  To eliminate rows that have been deleted from the result, a query must do extra processing. If 10% or more of the total rows in a table have been deleted, the performance of a query on the table degrades. However, your experience may vary depending on the size of the table, the table definition, and the query. If a table has a large number of deleted rows, consider purging those rows to improve performance. For more information on purging, see Purging Deleted Data.

- **Recovery performance**—Recovery is the action required for a cluster to restore K-safety after a crash. Large numbers of deleted records can degrade the performance of a recovery. To improve recovery performance, purge the deleted rows. For more information on purging, see Purging Deleted Data.

- **Concurrency**—DELETE and UPDATE take exclusive locks on the table. Only one DELETE or UPDATE transaction on a table can be in progress at a time and only when no loads (or INSERTs) are in progress. DELETEs and UPDATEs on different tables can be run concurrently.

  For detailed tips about improving DELETE and UPDATE performance, see DELETE and UPDATE Optimization.

Caution: Vertica does not remove deleted data immediately but keeps it as history for the purposes of historical query. A large amount of history can result in slower query performance.
performance. For information about how to configure the appropriate amount of history to retain, see Purging Deleted Data.

## DELETE and UPDATE Optimization

The process of optimizing DELETE and UPDATE queries is the same for both operations. Some simple steps can increase the query performance by tens to hundreds of times. The following sections describe several ways to improve projection design and improve DELETE and UPDATE queries to significantly increase DELETE and UPDATE performance.

**Note:** For large bulk deletion, Vertica recommends using Partitioned Tables where possible because they provide the best DELETE performance and improve query performance.

### Projection Column Requirements for Optimized Deletes

When all columns required by the DELETE or UPDATE predicate are present in a projection, the projection is optimized for DELETEs and UPDATEs. DELETE and UPDATE operations on such projections are significantly faster than on non-optimized projections.

For example, consider the following table and projections:

```sql
=> CREATE TABLE t (a INTEGER, b INTEGER, c INTEGER);
=> CREATE PROJECTION p1 (a, b, c) AS SELECT * FROM t ORDER BY a;
=> CREATE PROJECTION p2 (a, c) AS SELECT a, c FROM t ORDER BY c, a;
```

In the following query, both p1 and p2 are eligible for DELETE and UPDATE optimization because column a is available:

```sql
=> DELETE from t WHERE a = 1;
```

In the following example, only projection p1 is eligible for DELETE and UPDATE optimization because the b column is not available in p2:

```sql
=> DELETE from t WHERE b = 1;
```

### Optimized Deletes in Subqueries

To be eligible for DELETE optimization, all target table columns referenced in a DELETE or UPDATE statement's WHERE clause must be in the projection definition.

For example, the following simple schema has two tables and three projections:
CREATE TABLE tb1 (a INT, b INT, c INT, d INT);
CREATE TABLE tb2 (g INT, h INT, i INT, j INT);

The first projection references all columns in tb1 and sorts on column a:

CREATE PROJECTION tb1_p AS SELECT a, b, c, d FROM tb1 ORDER BY a;

The buddy projection references and sorts on column a in tb1:

CREATE PROJECTION tb1_p_2 AS SELECT a FROM tb1 ORDER BY a;

This projection references all columns in tb2 and sorts on column i:

CREATE PROJECTION tb2_p AS SELECT g, h, i, j FROM tb2 ORDER BY i;

Consider the following DML statement, which references tb1.a in its WHERE clause. Since both projections on tb1 contain column a, both are eligible for the optimized DELETE:

DELETE FROM tb1 WHERE tb1.a IN (SELECT tb2.i FROM tb2);

Restrictions

Optimized DELETEs are not supported under the following conditions:

- With replicated projections if subqueries reference the target table. For example, the following syntax is not supported:

  DELETE FROM tb1 WHERE tb1.a IN (SELECT e FROM tb2, tb2 WHERE tb2.e = tb1.e);

- With subqueries that do not return multiple rows. For example, the following syntax is not supported:

  DELETE FROM tb1 WHERE tb1.a = (SELECT k FROM tb2);

Projection Sort Order for Optimizing Deletes

Design your projections so that frequently-used DELETE or UPDATE predicate columns appear in the sort order of all projections for large DELETEs and UPDATEs.

For example, suppose most of the DELETE queries you perform on a projection look like the following:

DELETE from t where time_key < '1-1-2007'
To optimize the DELETEs, make time_key appear in the ORDER BY clause of all your projections. This schema design results in better performance of the DELETE operation.

In addition, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. For more information, see Combine RLE and Sort Order. To analyze projections for sort order issues, use the EVALUATE_DELETE_PERFORMANCE function.
Purging Deleted Data

In Vertica, delete operations do not remove rows from physical storage. `DELETE` marks rows as deleted, as does `UPDATE`, which combines delete and insert operations. In both cases, Vertica retains discarded rows as historical data, which remains accessible to historical queries until it is purged.

The cost of retaining historical data is twofold:

- Disk space is allocated to deleted rows and delete markers.
- Typical (non-historical) queries must read and skip over deleted data, which can impact performance.

A purge operation permanently removes historical data from physical storage and frees disk space for reuse. Only historical data that precedes the Ancient History Mark (AHM) is eligible to be purged.

You can purge data in two ways:

- **Set a purge policy.**
- **Manually purge data.**

In both cases, Vertica purges all historical data up to and including the AHM epoch and resets the AHM.

Caution: Large delete and purge operations can take a long time to complete, so use them sparingly. If your application requires deleting data on a regular basis, such as by month or year, consider designing tables that take advantage of table partitioning. If partitioning is not suitable, consider rebuilding the table.

Setting a Purge Policy

The preferred method for purging data is to establish a policy that determines which deleted data is eligible to be purged. Eligible data is automatically purged when the Tuple Mover performs mergeout operations.

Vertica provides two methods for determining when deleted data is eligible to be purged:
• Specifying the time for which delete data is saved

• Specifying the number of epochs that are saved

Specifying the Time for Which Delete Data Is Saved

Specifying the time for which delete data is saved is the preferred method for determining which deleted data can be purged. By default, Vertica saves historical data only when nodes are down.

To change the specified time for saving deleted data, use the HistoryRetentionTime configuration parameter:

```sql
=> ALTER DATABASE mydb SET HistoryRetentionTime = {seconds | -1};
```

In the above syntax:

• `seconds` is the amount of time (in seconds) for which to save deleted data.

• `-1` indicates that you do not want to use the HistoryRetentionTime configuration parameter to determine which deleted data is eligible to be purged. Use this setting if you prefer to use the other method (HistoryRetentionEpochs) for determining which deleted data can be purged.

The following example sets the history epoch retention level to 240 seconds:

```sql
=> ALTER DATABASE mydb SET HistoryRetentionTime = 240;
```

Specifying the Number of Epochs That Are Saved

Unless you have a reason to limit the number of epochs, Vertica recommends that you specify the time over which delete data is saved.

To specify the number of historical epoch to save through the HistoryRetentionEpochs configuration parameter:

1. Turn off the HistoryRetentionTime configuration parameter:

   ```sql
   => ALTER DATABASE mydb SET HistoryRetentionTime = -1;
   ```

2. Set the history epoch retention level through the HistoryRetentionEpochs configuration parameter:
ALTER DATABASE mydb SET HistoryRetentionEpochs = \{num_epochs | -1\};

- *num_epochs* is the number of historical epochs to save.
- -1 indicates that you do not want to use the HistoryRetentionEpochs configuration parameter to trim historical epochs from the epoch map. By default, HistoryRetentionEpochs is set to -1.

The following example sets the number of historical epochs to save to 40:

ALTER DATABASE mydb SET HistoryRetentionEpochs = 40;

Modifications are immediately implemented across all nodes within the database cluster. You do not need to restart the database.

**Note:** If both HistoryRetentionTime and HistoryRetentionEpochs are specified, HistoryRetentionTime takes precedence.

See [Epoch Management Parameters](#) for additional details.

### Disabling Purge

If you want to preserve all historical data, set the value of both historical epoch retention parameters to -1, as follows:

ALTER DATABASE mydb SET HistoryRetentionTime = -1;
ALTER DATABASE mydb SET HistoryRetentionEpochs = -1;

### Manually Purging Data

You manually purge deleted data as follows:

1. Determine the point in time to which you want to purge deleted data.
2. Set the ancient history mark (AHM) to this point in time using one of the following functions:
   - **SET_AHM_TIME** sets the AHM to the epoch that includes the specified TIMESTAMP value on the initiator node.
   - **SET_AHM_EPOCH** sets the AHM to the specified epoch.
- **GET_AHM_TIME** returns a TIMESTAMP value representing the AHM.
- **GET_AHM_EPOCH** returns the number of the epoch in which the AHM is located.
- **MAKE_AHM_NOW** sets the AHM to the greatest allowable value. This lets you purge all deleted data.

When you use SET_AHM_TIME or GET_AHM_TIME, keep in mind that the timestamp you specify is mapped to an epoch, which by default has a three-minute granularity. Thus, if you specify an AHM time of 2008-01-01 00:00:00.00, the resulting purge might permanently remove as much as the first three minutes of 2008, or fail to remove the last three minutes of 2007.

Note: The system prevents you from setting the AHM beyond the point where it prevents recovery in the event of node failure.

3. Purge deleted data from the desired projections with one of the following functions:
   - **PURGE** purges all projections in the physical schema.
   - **PURGE_TABLE** purges all projections anchored to the specified table.
   - **PURGE_PROJECTION** purges the specified projection.
   - **PURGE_PARTITION** purges a specified partition.

4. The tuple mover performs a mergeout operation to purge the data. Vertica periodically invokes the tuple mover to perform mergeout operations, as configured by tuple mover parameters. You can manually invoke the tuple mover by calling the function **DO_TM_TASK**.

   Caution: Manual purge operations can take a long time.

**Truncating Tables**

**TRUNCATE TABLE** removes all storage associated with the target table and its projections. Vertica preserves the table and the projection definitions. If the truncated table has out-of-date projections, those projections are cleared and marked up-to-date when TRUNCATE TABLE returns.

TRUNCATE TABLE commits the entire transaction after statement execution, even if truncating the table fails. You cannot roll back a TRUNCATE TABLE statement.
Use `TRUNCATE TABLE` for testing purposes. You can use it to remove all data from a table and load it with fresh data, without recreating the table and its projections.

**Table Locking**

`TRUNCATE TABLE` takes an O (owner) lock on the table until the truncation process completes. The savepoint is then released. If the operation cannot obtain an O lock on the target table, Vertica tries to close any internal Tuple Mover sessions that are running on that table. If successful, the operation can proceed. Explicit Tuple Mover operations that are running in user sessions do not close. If an explicit Tuple Mover operation is running on the table, the operation proceeds only when the operation is complete.

**Restrictions**

- You cannot truncate an external table.
- You can truncate the fact table of a pre-join projection. However, you cannot truncate the dimension table of the same pre-join projection. If a table acts as a dimension table in any pre-join projections, you must drop those projections before you can execute `TRUNCATE TABLE` on the dimension table.

**Examples**

```sql
=> INSERT INTO sample_table (a) VALUES (3);
=> SELECT * FROM sample_table;
a
---
3
(1 row)
=> TRUNCATE TABLE sample_table;
TRUNCATE TABLE
=> SELECT * FROM sample_table;
a
---
(0 rows)
```

**Rebuilding Tables**

You can reclaim disk space on a large scale by rebuilding tables, as follows:
1. Create a table with the same (or similar) definition as the table to rebuild.

2. Create projections for the new table.

3. Copy data from the target table into the new one with `INSERT .. SELECT`.

4. Drop the old table and its projections.

   Note: Rather than dropping the old table, you can rename it and use it as a backup copy. Before doing so, verify that you have sufficient disk space for both the new and old tables.

5. Rename the new table with `ALTER TABLE .. RENAME`, using the name of the old table.

   Caution: When you rebuild a table, Vertica purges the table of all delete vectors that precede the AHM. This prevents historical queries on any older epoch.

### Projection Considerations

- You must have enough disk space to contain the old and new projections at the same time. If necessary, you can drop some of the old projections before loading the new table. You must, however, retain at least one superprojection of the old table (or two buddy superprojections to maintain K-safety) until the new table is loaded. (See Prepare Disk Storage Locations in Installing Vertica for disk space requirements.)

- You can specify different names for the new projections or use `ALTER TABLE .. RENAME` to change the names of the old projections.

- The relationship between tables and projections does not depend on object names. Instead, it depends on object identifiers that are not affected by rename operations. Thus, if you rename a table, its projections continue to work normally.

### Dropping Tables

`DROP TABLE` drops a table from the database catalog. If any projections are associated with the table, `DROP TABLE` returns an error message unless it also includes the CASCADE option. One exception applies: the table only has an auto-generated superprojection (auto-projection) associated with it.
Using CASCADE

In the following example, DROP TABLE tries to remove a table that has several projections associated with it. Because it omits the CASCADE option, Vertica returns an error:

```sql
=> DROP TABLE d1;
NOTICE: Constraint - depends on Table d1
NOTICE: Projection d1p1 depends on Table d1
NOTICE: Projection d1p2 depends on Table d1
NOTICE: Projection d1p3 depends on Table d1
NOTICE: Projection f1d1p1 depends on Table d1
NOTICE: Projection f1d1p2 depends on Table d1
NOTICE: Projection f1d1p3 depends on Table d1
ERROR: DROP failed due to dependencies: Cannot drop Table d1 because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
=> DROP TABLE d1 CASCADE;
DROP TABLE
=> CREATE TABLE mytable (a INT, b VARCHAR(256));
CREATE TABLE
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

The next attempt includes the CASCADE option and succeeds:

```sql
=> DROP TABLE d1 CASCADE;
DROP TABLE
=> CREATE TABLE mytable (a INT, b VARCHAR(256));
CREATE TABLE
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

Using IF EXISTS

In the following example, DROP TABLE includes the option IF EXISTS. This option specifies not to report an error if one or more of the tables to drop does not exist. This clause is useful in SQL scripts—for example, to ensure that a table is dropped before you try to recreate it:

```sql
=> DROP TABLE IF EXISTS mytable;
DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Table doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```
Dropping and Restoring View Tables

Views that reference a table that is dropped and then replaced by another table with the same name continue to function and use the contents of the new table. The new table must have the same column definitions.
Managing Client Connections

You can limit the number of active concurrent sessions a user can open to the server. Limiting user sessions lets you better manage sessions and prevents excessive server communication.

You can define connection limits at three levels: user level, for an individual node or entire cluster.

You can use the following parameters to close a session that has been idle for a period of time:

- **IDLESESSIONTIMEOUT**: Sets how much time can elapse before a session times out.
- **DEFAULTIDLESESSIONTIMEOUT**: A configuration parameter that sets the default timeout value for users who do not have idlesessiontimeout set. See Security Parameters.

The IDLESESSIONTIMEOUT parameter applies only to sessions that are idle, not to sessions where query execution is in progress. An idle session is one that has no queries running. If a client is slow or unresponsive during query execution, that time does not apply to timeout. For example, if you are doing a streaming batch insert, the time that it takes to perform this operation is not counted towards timeout.

Use Cases

A user executes a query, and for some reason the query takes an unusually long time to finish (for example, because of server traffic or query complexity). In this case, the user may mistakenly think the query failed and opens another session to run the same query. Now, two sessions are running the same query, using an extra connection.

To prevent this situation, limit the number of sessions a user can run by creating or modifying a user and setting MAXCONNECTIONS. For example:

```sql
=> CREATE USER u1 MAXCONNECTIONS 10 ON NODE;
```

Another issue setting client connections prevents is when a user connects to the server many times. Too many user connections exhausts the number of allowable connections set with the MaxClientSessions parameter (see Managing Sessions).

**Note**: No user can have a MAXCONNECTIONS limit greater than the MaxClientSessions setting.
Manage Client Sessions

After setting a user's connection limits, you can manage the client sessions by monitoring changes to the user's connection parameters. Also be aware that behavior changes can occur with client connection limits when a node:

- Is removed
- Is added
- Goes down
- Comes up

Client Connection Changes

When you modify a user's client connection parameters, be aware that:

- Changing a user's MAXCONNECTIONS setting only affects new connections.
- Changing a user's connection mode from DATABASE to NODE does not affect current sessions. All new sessions are reserved on the invoking node rather than the entire cluster.
- Changing a user's connection mode from NODE to DATABASE does not affect current sessions. New connection requests are reserved on the entire cluster.

Node Going Up or Down

In terms of honoring connection limits, no significant impact exists when nodes go down or come up in between connection requests. No special actions are needed to handle this. However, note the following:

- If a node goes down, its active session exits and other nodes in the cluster also drop their sessions. This frees up connections. The query may hang in which case the blocked sessions are reasonable and as expected.
User-Defined Session Timeout

As the DBADMIN, you can set a user’s idle session timeout with IDLESESSIONTIMEOUT. A user can edit the session timeout value as follows:

`=> SET SESSION IDLESESSIONTIMEOUT 'interval';`

In the user role, you can only lower the IDLESESSIONTIMEOUT value. If you attempt to increase the value, the following error message appears:

```
ERROR 0: New idlesessiontimeout 00:12 would exceed user limit of 00:10
```

Run the following query to see the current IDLESESSIONTIMEOUT value:

```
=> SHOW IDLESESSIONTIMEOUT;
name | idlesessiontimeout
--------|------------------
user1 | 00:08
```

Set the DEFAULTIDLESESSIONTIMEOUT configuration parameter as the default session timeout value:

```
=> ALTER DATABASE <dbname> SET DEFAULTIDLESESSIONTIMEOUT = interval;
```

This value applies to users who do not have an IDLESESSIONTIMEOUT value set.

In the user role, you can lower the idle timeout for a specific session as follows:

```
=> SET SESSION IDLESESSIONTIMEOUT 'interval';
```

Close a Session for a User

If necessary, you can manually close a user session with CLOSE_USER_SESSIONS:

```
=> SELECT CLOSE_USER_SESSIONS ('user-name');
close_user_sessions
-----------------------------
Session close command sent. Check v_monitor.session for progress
```
Set Client Connection Limits

Setting a user's client connection limits allows you to control how many simultaneous sessions a user can run. This prevents Vertica from exhausting the allowable concurrent connections set by MaxClientSessions and improves the usability of the system.

MaxClientSessions Limits

Client connection limits cannot be higher than the limits set in MaxClientSessions. Therefore:

- If you need to increase a user's allowed concurrent connections, check to see if MaxClientSessions needs to be increased first.

- If you decrease MaxClientSessions, the database uses the lesser of the two connection values between MaxClientSessions and the user-level connection limit set with MAXCONNECTIONS.

Set Connection Limits for a User

Set connection limits for a user as follows:

```sql
=> CREATE USER user1 MAXCONNECTIONS 10 ON NODE IDLESESSIONTIMEOUT '10 mins';
```

This example limits user1 to ten concurrent connections on an individual node. An idle session for the user times out after ten minutes. For more information see CREATE USER.

You can also modify an existing user to add connection limits with ALTER USER.

```sql
=> ALTER USER user2 MAXCONNECTIONS 5 ON DATABASE;
```

View User Connection Limits and Session Timeouts

See the USERS table to determine if a user has connection limits and session timeouts implemented.

```sql
=> SELECT user_name, max_connections, connection_limit_mode, idle_session_timeout FROM users;
```

<table>
<thead>
<tr>
<th>user_name</th>
<th>max_connections</th>
<th>connection_limit_mode</th>
<th>idle_session_timeout</th>
</tr>
</thead>
</table>
IDLESESSIONTIMEOUT

When setting the IDLESESSIONTIMEOUT value, be aware that a system is considered idle when it is waiting for instructions from the client.

The system starts tracking the timeout value from the point the server starts waiting for any type of message. For example, if the server receives a parse and is waiting for a bind, the time spent waiting is considered idle time and IDLESESSIONTIMEOUT starts.

When you change a user's IDLESESSIONTIMEOUT value, the new value affects only those sessions started after the update.
Working with Projections

Unlike traditional databases that store data in tables, Vertica physically stores table data in projections, which are collections of table columns.

Projections store data in a format that optimizes query execution. Similar to materialized views, they store result sets on disk rather than compute them each time they are used in a query. Vertica automatically refreshes these result sets with updated or new data.

Projections can be generally classified in two ways:

- **How projection data is distributed** on the cluster. A projection can be defined to divide its data into multiple segments, or maintain all projection data as a single unsegmented unit.

- **How much data a projection contains, and the nature of that data.** For example, each table Vertica requires a superprojection, which contains all table columns. You can also create query-specific projections, which contain only the subset of table columns to process a given query.

For more general information about Vertica projections, see Physical Schema in Vertica Concepts.

Projection Naming

Vertica identifies projections according to the following conventions, where `proj-basename` is the name assigned to this projection by `CREATE PROJECTION`.

Unsegmented Projections

Unsegmented projections conform to the following naming conventions:

<table>
<thead>
<tr>
<th><code>table-name</code>_super</th>
<th>Identifies the auto projection that Vertica automatically creates when you initially load data into an unsegmented table, where <code>table-basename</code> is the table name specified in <code>CREATE TABLE</code>. The auto projection is always a superprojection. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>store.customer_dimension_super</code></td>
<td></td>
</tr>
</tbody>
</table>
proj-basename
[._unseg]

Identifies an unsegmented projection. If proj-basename is identical to the anchor table name, Vertica appends the string _unseg to the projection name. If the projection is copied on all nodes, this projection name maps to all instances.

For example:

store.customer_dimension_unseg

Segmented Projections

Segmented projections conform to the following naming convention:

proj-basename_
 offsetset

Identifies buddy projections for a segmented projection, where offset identifies the projection's node location relative to all other buddy projections. All buddy projections share the same project base name.

For example:

store.store_orders_fact_b0
store.store_orders_fact_b1

One exception applies: Vertica uses the following convention to name live aggregate projections:

- proj-basename
- proj-basename_b1
- ...

Projections of Copied Tables

Vertica creates projections for tables that you create from existing tables with CREATE TABLE LIKE...INCLUDING PROJECTIONS.

The following table describes the algorithm that Vertica uses to assign base names to the new projections:
### Source projection name | Target
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Projection base name</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Same</td>
<td>Different</td>
</tr>
<tr>
<td>string</td>
<td>Different</td>
<td>Same</td>
</tr>
<tr>
<td>string</td>
<td>Different</td>
<td>Different</td>
</tr>
<tr>
<td>srcTableName_string</td>
<td>Same</td>
<td>Different</td>
</tr>
<tr>
<td>srcTableName_string</td>
<td>Different</td>
<td>Different</td>
</tr>
</tbody>
</table>

For example, given the following table and its segmented projections:

<table>
<thead>
<tr>
<th>Table name</th>
<th>Projection names</th>
</tr>
</thead>
</table>
| public.T1 | T1_b0  
T1_b1  
T1_proj1_b0  
T1_proj1_b1  
proj2_b0  
proj2_b1 |

The following `CREATE TABLE`...LIKE statements copy table `public.T1` and its projections to tables `public.T2` and `private.T1`:

```sql
CREATE TABLE public.T2 LIKE public.T1 INCLUDING PROJECTIONS;
CREATE TABLE private.T1 LIKE public.T1 INCLUDING PROJECTIONS;
```

For each new table, Vertica copies the `public.T1` projections as follows:

<table>
<thead>
<tr>
<th>New table name</th>
<th>Copied projection names</th>
</tr>
</thead>
</table>
| public.T2 | T2_b0  
T2_b1  
T2_proj1_b0  
T2_proj1_b1  
T2_proj2_b0  
T2_proj2_b1 |
| private.T1 | T1_b0  
T1_b1  
T1_proj1_b0  
T1_proj1_b1  
proj2_b0  
proj2_b1 |
Auto-Projections

Auto-projections are superprojections that Vertica automatically generates for tables, both temporary and persistent. The following rules apply to all auto-projections:

- Vertica creates the auto-projection in the same schema as the table.
- Auto-projections conform to encoding, sort order, segmentation, and K-safety as specified in the table's creation statement.
- If the table creation statement contains an AS SELECT clause, Vertica uses some properties of the projection definition's underlying query.

Auto-Projection Triggers

The conditions for creating auto-projections differ, depending on whether the table is temporary or persistent:

<table>
<thead>
<tr>
<th>Table type</th>
<th>Auto-projection trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporary</td>
<td>CREATE TEMPORARY TABLE statement unless it includes NO PROJECTION.</td>
</tr>
<tr>
<td>Persistent</td>
<td>CREATE TABLE statement contains one of these clauses:</td>
</tr>
<tr>
<td></td>
<td>• AS SELECT</td>
</tr>
<tr>
<td></td>
<td>• ENCODED BY</td>
</tr>
<tr>
<td></td>
<td>• ORDER BY</td>
</tr>
<tr>
<td></td>
<td>• SEGMENTED BY / UNSEGMENTED</td>
</tr>
<tr>
<td></td>
<td>• KSAFE</td>
</tr>
<tr>
<td></td>
<td>If none of these conditions is true, Vertica automatically creates a superprojection</td>
</tr>
<tr>
<td></td>
<td>(if one does not already exist) only when you first load data into the table with</td>
</tr>
<tr>
<td></td>
<td>INSERT or COPY.</td>
</tr>
</tbody>
</table>
Default Segmentation and Sort Order

If `CREATE TABLE` or `CREATE TEMPORARY TABLE` omits a segmentation (`SEGMENTED BY` or `UNSEGMENTED`) or `ORDER BY` clause, Vertica segments and sorts auto-projections according to the table's manner of creation:

- If `CREATE [TEMPORARY] TABLE` contains an `AS SELECT` clause, and the query output is segmented, the auto-projection uses the same segmentation. If the result set is already sorted, the projection uses the same sort order.

- In all other cases, Vertica evaluates table column constraints to determine how to sort and segment the projection, as shown below:

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Sorted by:</th>
<th>Segmented by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>Primary key</td>
<td>Primary key</td>
</tr>
<tr>
<td>Primary and foreign keys</td>
<td>1. Foreign keys</td>
<td>Primary key</td>
</tr>
<tr>
<td></td>
<td>2. Primary key</td>
<td></td>
</tr>
<tr>
<td>Foreign keys only</td>
<td>1. Foreign keys</td>
<td>All columns excluding <code>LONG data types</code></td>
</tr>
<tr>
<td></td>
<td>2. Remaining columns excluding <code>LONG data types</code>, in the order specified by <code>CREATE TABLE</code></td>
<td>up to the limit set in configuration parameter <code>MaxAutoSegColumns</code> (by default 32).</td>
</tr>
<tr>
<td>None</td>
<td>All columns excluding <code>LONG data types</code>, in the order specified by <code>CREATE TABLE</code>.</td>
<td>1. Small (&lt; 8 byte) data type columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Large (&gt;8 byte) data type columns</td>
</tr>
</tbody>
</table>

**Unsegmented Projections**

In many cases, dimension tables are relatively small, so you do not need to segment them. Accordingly, you should design a K-safe database so projections for its dimension tables are
replicated without segmentation on all cluster nodes. You create unsegmented projections with a CREATE PROJECTION statement that includes the clause UNSEGMENTED ALL NODES. This clause specifies to create identical instances of the projection on all cluster nodes.

The following example shows how to create an unsegmented projection for the table store.store_dimension:

```
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
   AS SELECT store_key, store_name, store_city, store_state
   FROM store.store_dimension
   UNSEGMENTED ALL NODES;
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, store.store_dimension_proj. The keyword ALL NODES specifies to replicate the projection on all nodes:

```
=> \dj store.store_dimension_proj
```

For more information about projection name conventions, see Projection Naming.

### Segmented Projections

You typically create segmented projections for large fact tables. Vertica splits segmented projections into chunks (segments) of similar size and distributes these segments evenly across the cluster. System K-safety determines how many duplicates (buddies) of each segment are created and maintained on different nodes.

You create segmented projections with a CREATE PROJECTION statement that includes a SEGMENTED BY clause.

The following CREATE PROJECTION statement creates projection public.employee_dimension_super. It specifies to include all columns in table public.employee_dimension. The hash segmentation clause invokes the Vertica HASH function to segment projection data on the column employee_key; it also includes the ALL NODES clause, which specifies to distribute projection data evenly across all nodes in the cluster:

```
=> CREATE PROJECTION public.employee_dimension_super
   AS SELECT * FROM public.employee_dimension
   ORDER BY employee_key
```
If the database is K-safe, Vertica creates multiple buddies for this projection and distributes them on different nodes across the cluster. In this case, database K-safety is set to 1, so Vertica creates two buddies for this projection. It uses the projection name `employee_dimension_super` as the basename for the two buddy identifiers it creates—in this example, `employee_dimension_super_b0` and `employee_dimension_super_b1`:

```
=> SELECT projection_name FROM projections WHERE projection_basename='employee_dimension_super';
  projection_name
-------------------
  employee_dimension_super_b0
  employee_dimension_super_b1
(2 rows)
```

### K-Safe Database Projections

K-safety is implemented differently for segmented and unsegmented projections, as described below. Examples assume database K-safety is set to 1 in a 3-node database, and uses projections for two tables:

- `store.store_orders_fact` is a large fact table. The projection for this table should be segmented. Vertica distributes projection segments uniformly across the cluster.
- `store.store_dimension` is a smaller dimension table. The projection for this table should be unsegmented. Vertica copies a complete instance of this projection on each cluster node.

### Segmented Projections

In a K-safe database, the database requires K+1 instances, or buddies, of each projection segment. For example, if database K-safety is set to 1, the database requires two instances, or buddies, of each projection segment.

You can set K-safety on individual segmented projections through the `CREATE PROJECTION` option `KSAFE`. Projection K-safety must be equal to or greater than database K-safety. If you omit setting `KSAFE`, the projection obtains K-safety from the database.

The following `CREATE PROJECTION` defines a segmented projection for the fact table `store.store_orders_fact`: 

```sql
SEGMENTED BY hash(employee_key) ALL NODES;
```
The following keywords in the CREATE PROJECTION statement pertain to setting projection K-safety:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGMENTED BY</td>
<td>Specifies how to segment projection data for distribution across the cluster. In this example, the segmentation expression specifies Vertica's built-in HASH function.</td>
</tr>
<tr>
<td>ALL NODES</td>
<td>Specifies to distribute projection segments across all cluster nodes.</td>
</tr>
<tr>
<td>K-SAFE 1</td>
<td>Sets K-safety to 1. Vertica creates two projection buddies with these identifiers:</td>
</tr>
<tr>
<td></td>
<td>• store.store_orders_fact_b0</td>
</tr>
<tr>
<td></td>
<td>• store.store_orders_fact_b1</td>
</tr>
</tbody>
</table>

### Unsegmented Projections

In a K-safe database, unsegmented projections must be replicated on all nodes. Thus, the CREATE PROJECTION statement for an unsegmented projection must include the segmentation clause UNSEGMENTED ALL NODES. This instructs Vertica to create identical instances (buddies) of the projection on all cluster nodes. If you create an unsegmented projection on a single node, Vertica regards it unsafe and does not use it.

The following example shows how to create an unsegmented projection for the table store.store_dimension:

```sql
=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
   AS SELECT store_key, store_name, store_city, store_state
   FROM store.store_dimension
   UNSEGMENTED ALL NODES;
CREATE PROJECTION
```

Vertica uses the same name to identify all instances of the unsegmented projection—in this example, store.store_dimension_proj. The keyword ALL NODES specifies to replicate the projection on all nodes:
=> \dj store.store_dimension_proj

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Owner</th>
<th>Node</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>dbadmin</td>
<td>v_vmart_node0001</td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>dbadmin</td>
<td>v_vmart_node0002</td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>dbadmin</td>
<td>v_vmart_node0003</td>
<td></td>
</tr>
</tbody>
</table>

(3 rows)

For more information about projection name conventions, see Projection Naming.

**Refreshing Projections**

When you create a projection for a table that already contains data, Vertica does not automatically load that data into the new projection. Instead, you must explicitly refresh that projection. Until you do so, the projection cannot participate in executing queries on its anchor table.

You can refresh a projection with one of the following functions:

- **START_REFRESH** refreshes projections in the current schema with the latest data of their respective anchor tables. START_REFRESH runs asynchronously in the background.

- **REFRESH** synchronously refreshes one or more table projections in the foreground.

Both functions update system tables that maintain information about a projection's refresh status: PROJECTION_REFRESHES, PROJECTIONS, and PROJECTION_CHECKPOINT_EPOCHS.

**Getting Projection Refresh Information**

You can query PROJECTION_REFRESHES and PROJECTIONS to view the progress of the refresh operation. You can also call the Vertica function GET_PROJECTIONS to view the final status of projection refreshes for a given table:

```sql
=> SELECT GET_PROJECTIONS('customer_dimension');

GET_PROJECTIONS

---------------------
-----
Current system K is 1.
# of Nodes: 3.
Table public.customer_dimension has 2 projections.

Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]
```
Refresh Methods

Vertica can refresh a projection from one of its buddies, if one is available. In this case, the target projection gets the source buddy's historical data. Otherwise, the projection is refreshed from scratch with data of the latest epoch at the time of the refresh operation. In this case, the projection cannot participate in historical queries on any epoch that precedes the refresh operation.

To determine the method used to refresh a given projection, query REFRESH_METHOD from system table PROJECTION_REFRESHES.

Dropping Projections

Projections can be dropped explicitly through the DROP PROJECTION statement. They are also implicitly dropped when you drop their anchor table.
Partitioning Tables

Data partitioning is defined as a table property, and is implemented on all projections of that table. On all load, refresh, and recovery operations, the Vertica Tuple Mover automatically partitions data into separate ROS containers. Each ROS container contains data for a single partition or partition group; depending on space requirements, a partition or partition group can span multiple ROS containers.

For example, it is common to partition data by time slices. If a table contains decades of data, you can partition it by year. If the table contains only one year of data, you can partition it by month.

Logical divisions of data can significantly improve query execution. For example, if you query a table on a column that is in the table's partition clause, the query optimizer can quickly isolate the relevant ROS containers (see Partition Pruning).

Partitions can also facilitate DML operations. For example, given a table that is partitioned by months, you might drop all data for the oldest month when a new month begins. In this case, Vertica can easily identify the ROS containers that store the partition data to drop. For details, see Managing Partitions.

Defining Partitions

You can specify partitioning for new and existing tables:

- Define partitioning for a table with CREATE TABLE.
- Specify partitioning for an existing table by modifying its definition with ALTER TABLE.
- Create partition groups to consolidate partitions into logical subsets, minimizing the use of ROS storage.

Partitioning a New Table

Use CREATE TABLE to partition a new table, as specified by the PARTITION BY clause:

```
CREATE TABLE table-name... PARTITION BY partition-expression [ GROUP BY group-expression ] [ REORGANIZE ];
```
The following statements create the `store_orders` table and load data into it. The `CREATE TABLE` statement includes a simple `partition clause` that specifies to partition data by year:

```
=> CREATE TABLE public.store_orders
(  
  order_no int,
  order_date timestamp NOT NULL,
  shipper varchar(20),
  ship_date date  
)
UNSEGMENTED ALL NODES
PARTITION BY YEAR(order_date);
CREATE TABLE
=> COPY store_orders FROM '/home/dbadmin/export_store_orders_data.txt' DIRECT;
41834
```

This `COPY` statement specifies `DIRECT`, so Vertica loads the new table data directly into ROS storage. As it does so, the Tuple Mover executes this table's partition clause by dividing yearly order into separate partitions.

**Note:** The Tuple Mover regularly executes `moveout` and `mergeout` operations, which respectively move new data from WOS into ROS, and consolidate ROS containers. If a table specifies partitioning, the Tuple Mover merges its data into ROS containers accordingly.

In this case, the Tuple Mover creates four partition keys for the loaded data—2017, 2016, 2015, and 2014—and divides the data into separate ROS containers accordingly:

```
=> SELECT dump_table_partition_keys('store_orders');
...
Partition keys on node v_vmart_node0001
  Projection 'store_orders_unseg_super'
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: 2017
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: 2016
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: 2015
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: 2014
Partition keys on node v_vmart_node0002
  Projection 'store_orders_unseg_super'
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: 2017
...
(1 row)
```
As new data is loaded into store_orders, the Tuple Mover merges it into the appropriate partitions, creating partition keys as needed for new years.

Partitioning Existing Table Data

Use **ALTER TABLE** to partition or repartition an existing table, as specified by the **PARTITION BY** clause:

```
ALTER TABLE table-name PARTITION BY partition-expression [ GROUP BY group-expression ] [ REORGANIZE ];
```

For example, you might repartition the store_orders table, defined earlier. The following **ALTER TABLE** divides all store_orders data into monthly partitions for each year, each partition key identifying the order date year and month:

```
=> ALTER TABLE store_orders
    PARTITION BY EXTRACT(YEAR FROM order_date)*100 + EXTRACT(MONTH FROM order_date)
    GROUP BY EXTRACT(YEAR FROM order_date)*100 + EXTRACT(MONTH FROM order_date);
NOTICE 8364: The new partitioning scheme will produce partitions in 42 physical storage containers per projection
WARNING 6100: Using PARTITION expression that returns a Numeric value
HINT: This PARTITION expression may cause too many data partitions. Use of an expression that returns a more accurate value, such as a regular VARCHAR or INT, is encouraged
WARNING 4493: Queries using table "store_orders" may not perform optimally since the data may not be repartitioned in accordance with the new partition expression
HINT: Use "ALTER TABLE public.store_orders REORGANIZE;" to repartition the data
```

After executing this statement, Vertica drops existing partition keys. However, the partition clause omits **REORGANIZE**, so existing data remains stored according to the previous partition clause. This can put table partitioning in an inconsistent state and adversely affect query performance, **DROP_PARTITIONS**, and node recovery. In this case, you must explicitly request Vertica to reorganize existing data into new partitions, in one of the following ways:

- **Issue ALTER TABLE…REORGANIZE:**

  ```
  ALTER TABLE table-name REORGANIZE;
  ```

- **Call the Vertica meta-function PARTITION_TABLE.**

For example:

```
=> ALTER TABLE store_orders REORGANIZE;
NOTICE 4785: Started background repartition table task
ALTER TABLE
```

**ALTER TABLE…REORGANIZE and PARTITION_TABLE** operate identically: both split any ROS containers where partition keys do not conform with the new partition clause. On
executing its next mergeout, the Tuple Mover merges partitions into the appropriate ROS containers.

**Partition Grouping**

Partition groups consolidate partitions into logical subsets that minimize use of ROS storage. Reducing the number of ROS containers to store partitioned data helps facilitate DML operations such as DELETE and UPDATE, and avoid ROS pushback. For example, you can group date partitions by year. By doing so, the Tuple Mover allocates ROS containers for each year group, and merges individual partitions into these ROS containers accordingly.

**Creating Partition Groups**

You create partition groups by qualifying the PARTITION BY clause with a GROUP BY clause:

```sql
ALTER TABLE table-name PARTITION BY partition-expression [ GROUP BY group-expression ]
```

The GROUP BY clause specifies how to consolidate partition keys into groups, where each group is identified by a unique partition group key. For example, the following ALTER TABLE statement specifies to repartition the store_orders table (shown in Partitioning a New Table) by order dates, grouping partition keys by year. The group expression—DATE_TRUNC ('year', (order_date)::DATE)—uses the partition expression order_date::DATE to generate partition group keys:

```sql
=> ALTER TABLE store_orders
        PARTITION BY order_date::DATE BY DATE_TRUNC('year', (order_date)::DATE) REORGANIZE;
NOTICE 8364: The new partitioning scheme will produce partitions in 4 physical storage containers per projection
NOTICE 4785: Started background repartition table task
```

In this case, the order_date column dates span four years. The Tuple Mover creates four partition group keys, and merges store_orders partitions into group-specific ROS storage containers accordingly:

```sql
=> SELECT DUMP_TABLE_PARTITION_KEYS('store_orders');
...
Partition keys on node v_vmart_node0001
  Projection 'store_orders_unseg_super'
  Storage [ROS container]
    No of partition keys: 173
    Partition keys: 2017-01-02 2017-01-03 2017-01-04 ... 2017-09-25 2017-09-26 2017-09-27
  Storage [ROS container]
    No of partition keys: 212
```
No of partition keys: 213
2015-11-26 2015-11-27
Storage [ROS container]
No of partition keys: 211
Projection 'store_orders_unseg_super'
Storage [ROS container]
No of partition keys: 173

Caution: This example demonstrates how partition grouping can facilitate more efficient use of ROS storage. However, grouping all partitions into several large and static ROS containers can adversely affect performance, especially for a table that is subject to frequent DML operations. Frequent load operations in particular can incur considerable merge overhead, which, in turn, reduces performance.

Vertica recommends that you use `CALENDAR_HIERARCHY_DAY`, as a partition clause’s group expression. This function automatically groups DATE partition keys into a dynamic hierarchy of years, months, and days. Doing so helps minimize merge-related issues. For details, see Hierarchical Partitioning.

Managing Partitions Within Groups

You can use various partition management functions, such as `DROP_PARTITIONS` or `MOVE_PARTITIONS_TO_TABLE`, to target a range of order dates within a given partition group, or across multiple partition groups. In the previous example, each group contains partition keys of different dates within a given year. You can use `DROP_PARTITIONS` to drop order dates that span two years, 2014 and 2015:

```
```

Important: The drop operation requires Vertica to split the ROS containers that store partition groups for these two years. To do so, the function's `force_split` parameter must be set to true.

Hierarchical Partitioning

The meta-function `CALENDAR_HIERARCHY_DAY` leverages partition grouping. You specify this function as the partitioning `GROUP BY` expression. `CALENDAR_HIERARCHY_DAY` organizes a table’s date partitions into a hierarchy of groups: the oldest date partitions are grouped by
year, more recent partitions are grouped by month, and the most recent date partitions
remain ungrouped. Grouping is dynamic: as recent data ages, the Tuple Mover merges their
partitions into month groups, and eventually into year groups.

Managing Timestamped Data

Partition consolidation strategies are especially important for managing timestamped data,
where the number of partitions can quickly escalate and risk ROS pushback. For example, the
following statements create the store_orders table and load data into it. The
CREATE TABLE statement includes a simple partition clause that specifies to partition data by
date:

```sql
=> DROP TABLE IF EXISTS public.store_orders CASCADE;
=> CREATE TABLE public.store_orders
    (order_no int,
     order_date timestamp NOT NULL,
     shipper varchar(20),
     ship_date date
    )
UNSEGMENTED ALL NODES PARTITION BY order_date::DATE;
CREATE TABLE
=> COPY store_orders FROM '/home/dbadmin/export_store_orders_data.txt' DIRECT;
41834
(1 row)
```

This COPY statement specifies DIRECT, so Vertica loads the new table data directly into ROS
storage. As it does so, it executes this table's partition clause by dividing daily orders into
separate partitions—in this case, 809 partitions, where each partition requires its own
ROS container:

```sql
=> SELECT COUNT(DISTINCT ros_id) NumROS, node_name FROM PARTITIONS
       WHERE projection_name ilike '%store_orders_super%'
    GROUP BY node_name ORDER BY node_name;
 NumROS | node_name
---------+---------------------
   809 | v_vmart_node0001
   809 | v_vmart_node0002
   809 | v_vmart_node0003
(3 rows)
```
This is far above the recommended maximum of 50 partitions per projection. This number is also close to the default system limit of 1024 ROS containers per projection, risking ROS pushback in the near future.

You can approach this problem in several ways:

- Consider consolidating table data into larger partitions—for example, partition by month instead of day. However, partitioning data at this level might limit effective use of partition management functions.

- Regularly archive older partitions, and thereby minimize the number of accumulated partitions. However, this requires an extra layer of data management, and also inhibits access to historical data.

Alternatively, you can use CALENDAR_HIERARCHY_DAY to automatically merge partitions into a date-based hierarchy of partition groups. Each partition group is stored in its own set of ROS containers, apart from other groups. You specify this function in the table partition clause as follows:

\[
\text{PARTITION BY} \quad \text{partition-expression} \\
\text{GROUP BY} \quad \text{CALENDAR_HIERARCHY_DAY(} \quad \text{partition-expression} \quad [, \quad \text{active-months}[\text{,}\quad \text{active-years}]] \quad \text{)}
\]

Important: Two requirements apply to using CALENDAR_HIERARCHY_DAY in a partition clause:

- \text{partition-expression} must be a \text{DATE}.

- The partition expressions specified by the PARTITION \text{ BY} clause and CALENDAR_HIERARCHY_DAY must be identical.
For example, given the previous table, you can repartition it as follows:

```sql
=> ALTER TABLE public.store_orders
  PARTITION BY order_date::DATE
  GROUP BY CALENDAR_HIERARCHY_DAY(order_date::DATE, 2, 2) REORGANIZE;
```

### Grouping DATE Data Hierarchically

CALENDAR_HIERARCHY_DAY creates hierarchies of partition groups, and merges partitions into the appropriate groups. It does so by evaluating the partition expression of each table row with the following algorithm, to determine its partition group key:

```sql
GROUP BY (CASE WHEN DATEDIFF('YEAR', partition-expression, NOW()::TIMESTAMPTZ(6)) >= active-years
  THEN DATE_TRUNC('YEAR', partition-expression::DATE)
  WHEN DATEDIFF('MONTH', partition-expression, NOW()::TIMESTAMPTZ(6)) >= active-months
  THEN DATE_TRUNC('MONTH', partition-expression::DATE)
  ELSE DATE_TRUNC('DAY', partition-expression::DATE) END);
```

In this example, the algorithm compares order_date in each store_orders row to the current date as follows:

1. Determines whether order_date is in an inactive year.
   
   If order_date is in an inactive year, the row's partition group key resolves to that year. The row is merged into a ROS container for that year.

2. If order_date is an active year, CALENDAR_HIERARCHY_DAY evaluates order_date to determine whether it is in an inactive month.
   
   If order_date is in an inactive month, the row's partition group key resolves to that month. The row is merged into a ROS container for that month.

3. If order_date is in an active month, the row's partition group key resolves to the order_date day. This row is merged into a ROS container for that day. Any rows where order_date is a future date is treated in the same way.

**Important:** The CALENDAR_HIERARCHY_DAY algorithm assumes that most table activity is focused on recent dates. Setting `active-years` and `active-months` to a low number \( \geq 2 \) serves to isolate most merge activity to date-specific containers, and incurs minimal overhead. Vertica recommends that you use the default setting of 2 for `active-years` and `active-months`. For most users, these settings achieve an optimal balance between ROS storage and performance.
For example, if the current date is 2017-09-26, CALENDAR_HIERARCHY_DAY resolves \textit{active-years} and \textit{active-months} to the following date spans:

- \textit{active-years}: 2016-01-01 to 2017-12-31. Partitions in active years are grouped into monthly ROS containers or are merged into daily ROS containers. Partitions from earlier years are regarded as inactive and merged into yearly ROS containers.

- \textit{active-months}: 2017-08-01 to 2017-09-30. Partitions in active months are merged into daily ROS containers.

Now, the total number of ROS containers is reduced to 40 per projection:

```sql
=> SELECT COUNT (DISTINCT ros_id) NumROS, node_name FROM PARTITIONS
       WHERE projection_name ilike '%store_orders_super%' GROUP BY node_name ORDER BY node_name;

<table>
<thead>
<tr>
<th>NumROS</th>
<th>node_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>40</td>
<td>v_vmart_node0002</td>
</tr>
<tr>
<td>40</td>
<td>v_vmart_node0003</td>
</tr>
</tbody>
</table>

(3 rows)
```

Note: Regardless of how the Tuple Mover groups and merges partitions, it always identifies one or more partitions or partition groups as active, as specified by configuration parameter \texttt{ActivePartitionCount}. For details, see \texttt{Active and Inactive Partitions}.  

Vertica Analytic Database (9.0.x)
Dynamic Regrouping

As shown earlier, CALENDAR_HIERARCHY_DAY references the current date when it creates partition group keys and merges partitions. As the calendar advances, the Tuple Mover reevaluates the partition group keys of tables that are partitioned with this function, and moves partitions as needed to different ROS containers.

Thus, given the previous example, on 2017-10-01 the Tuple Mover creates a monthly ROS container for August partitions. All partition keys between 2017-08-01 and 2017-08-31 are merged into the new ROS container 2017-08:

Likewise, on 2018-01-01, the Tuple Mover creates a ROS container for 2016 partitions. All partition keys between 2016-01-01 and 2016-12-31 that were previously grouped by month are merged into the new yearly ROS container:
Caution: After older partitions are grouped into months and years, any partition operation that acts on a subset of older partition groups, is liable to split ROS containers into smaller ROS containers for each partition—for example, `MOVE_PARTITIONS_TO_TABLE`, where `force-split` is set to true. These operations can lead to ROS pushback. If you anticipate frequent operations on hierarchically grouped partitions, consider modifying the partition expression so partitions are grouped no higher than months.

Customizing Partition Group Hierarchies

Vertica provides a single function, `CALENDAR_HIERARCHY_DAY`, to facilitate hierarchical partitioning. Vertica stores the `GROUP BY` clause as a CASE statement that you can edit to suit your own requirements.

For example, Vertica stores the `store_orders` partition clause as follows:

```sql
=\> ALTER TABLE public.store_orders
  PARTITION BY order_date::DATE
  GROUP BY CALENDAR_HIERARCHY_DAY(order_date::DATE, 2, 2);

=\> select export_tables('','store_orders');
...
CREATE TABLE public.store_orders ( ... )

PARTITION BY (store_orders.order_date::date)
GROUP BY (CASE WHEN ("datediff"('year', (store_orders.order_date)::date, ((now())::timestamptz(6)::date) >= 2)
THEN (date_trunc('year', (store_orders.order_date)::date))::date
```
WHEN ("datediff"('month', (store_orders.order_date)::date, ((now())::timestamptz)::date) >= 2)
    THEN (date_trunc('month', (store_orders.order_date)::date))::date
ELSE (store_orders.order_date)::date
END);

You can modify the CASE statement to customize the hierarchy of partition groups. For example, the following CASE statement creates a hierarchy of months, days, and hours:

=> ALTER TABLE store_orders
    PARTITION BY (store_orders.order_date)
    GROUP BY (CASE WHEN DATEDIFF('MONTH', store_orders.order_date, NOW())::TIMESTAMPZ(6)) >= 2
        THEN DATE_TRUNC('MONTH', store_orders.order_date::DATE)
        WHEN DATEDIFF('DAY', store_orders.order_date, NOW())::TIMESTAMPZ(6)) >= 2
        THEN DATE_TRUNC('DAY', store_orders.order_date::DATE)
        ELSE DATE_TRUNC('hour', store_orders.order_date::DATE) END);

**Partitioning and Segmentation**

In Vertica, partitioning and segmentation are separate concepts and achieve different goals to localize data:

- **Segmentation** refers to organizing and distributing data across cluster nodes for fast data purges and query performance. Segmentation aims to distribute data evenly across multiple database nodes so all nodes participate in query execution. You specify segmentation with the `CREATE PROJECTION` statement's `hash segmentation` clause.

- **Partitioning** specifies how to organize data within individual nodes for distributed computing. Node partitions let you easily identify data you wish to drop and help reclaim disk space. You specify partitioning with the `CREATE TABLE` statement's `PARTITION BY` clause.

For example: partitioning data by year makes sense for retaining and dropping annual data. However, segmenting the same data by year would be inefficient, because the node holding data for the current year would likely answer far more queries than the other nodes.

The following diagram illustrates the flow of segmentation and partitioning on a four-node database cluster:

1. Example table data
2. Data segmented by HASH(order_id)
3. Data segmented by hash across four nodes

4. Data partitioned by year on a single node

While partitioning occurs on all four nodes, the illustration shows partitioned data on one node for simplicity.

See Also

- Reclaiming Disk Space From Deleted Table Data
- Identical Segmentation
- Projection Segmentation
- CREATE PROJECTION
- CREATE TABLE

Managing Partitions

You can manage partitions with the following operations:
Drop partitions

Archive partitions

Swap partitions

Minimize partitions

View partition storage

Dropping Partitions

Use the DROP_PARTITIONS function to drop one or more partition keys for a given table. You can specify a single partition key or a range of partition keys. If the WOS contains data for the target table, DROP_PARTITIONS first invokes a moveout operation.

For example, the table shown in Partitioning a New Table is partitioned by column order_date:

```
=> CREATE TABLE public.store_orders
  
  (  
    order_no int,
    order_date timestamp NOT NULL,
    shipper varchar(20),
    ship_date date
  )
  PARTITION BY YEAR(order_date);
```

Given this table definition, Vertica creates a partition key for each unique order_date year—in this case, 2017, 2016, 2015, and 2014—and divides the data into separate ROS containers accordingly.

The following DROP_PARTITIONS statement drops from table store_orders all order records associated with partition key 2014:

```
=> SELECT DROP_PARTITIONS ('store_orders', 2014, 2014);
Partition dropped
```

Splitting Partition Groups

If a table partition clause includes a GROUP BY clause, partitions are consolidated in the ROS by their partition group keys. DROP_PARTITIONS can then specify a range of partition keys within a given partition group, or across multiple partition groups. In either case, the drop
operation requires Vertica to split the ROS containers that store these partitions. To do so, the function's `force_split` parameter must be set to true.

For example, the `store_orders` table shown above can be repartitioned with a `GROUP BY` clause as follows:

```
=> ALTER TABLE store_orders
    PARTITION BY order_date::DATE GROUP BY DATE_TRUNC('year', (order_date)::DATE) REORGANIZE;
```

With all 2014 order records having been dropped earlier, `order_date` values now span three years—2017, 2016, and 2015. Accordingly, the Tuple Mover creates three partition group keys for each year, and designates one or more ROS containers for each group. It then merges `store_orders` partitions into the appropriate groups.

The following `DROP_PARTITIONS` statement specifies to drop order dates that span two years, 2014 and 2015:

```
=> SELECT DROP_PARTITIONS('store_orders', '2015-05-30', '2016-01-16', 'true');
Partition dropped
```

The drop operation requires Vertica to drop partitions from two partition groups—2015 and 2016. These groups span at least two ROS containers, which must be split in order to remove the target partitions. Accordingly, the function's `force_split` parameter is set to true.

### Scheduling Partition Drops

If your hardware has fixed disk space, you might need to configure a regular process to roll out old data by dropping partitions.

For example, if you have only enough space to store data for a fixed number of days, configure Vertica to drop the oldest partition keys. To do so, create a time-based job scheduler such as `cron` to schedule dropping the partition keys during low-load periods.

If the ingest rate for data has peaks and valleys, you can use two techniques to manage how you drop partition keys:

- Set up a process to check the disk space on a regular (daily) basis. If the percentage of used disk space exceeds a certain threshold—for example, 80%—drop the oldest partition keys.

- Add an artificial column in a partition that increments based on a metric like row count. For example, that column might increment each time the row count increases by 100 rows. Set up a process that queries this column on a regular (daily) basis. If the value in the new column exceeds a certain threshold—for example, 100—drop the oldest partition keys, and set the column value back to 0.
Table Locking

DROP_PARTITIONS acquires an exclusive O lock on the target table to block any DML operation (DELETE, UPDATE, INSERT, or COPY) that might affect table data. The lock also blocks SELECT statements that are issued at SERIALIZABLE isolation level.

If the operation cannot obtain an O lock on the target table, Vertica tries to close any internal Tuple Mover sessions that are running on that table. If successful, the operation can proceed. Explicit Tuple Mover operations that are running in user sessions do not close. If an explicit Tuple Mover operation is running on the table, the operation proceeds only when the operation is complete.

Archiving Partitions

You can move partitions from one table to another with the Vertica function MOVE_PARTITIONS_TO_TABLE. This function is useful for archiving old partitions, as part of the following procedure:

1. Identify the partitions to archive, and move them to a temporary staging table with MOVE_PARTITIONS_TO_TABLE.

2. Back up the staging table.

3. Drop the staging table.

You restore archived partitions at any time.

Move Partitions to Staging Tables

You archive historical data by identifying the partitions you wish to remove from a table. You then move each partition (or group of partitions) to a temporary staging table.

Before calling MOVE_PARTITIONS_TO_TABLE:

- Drop any pre-join projections associated with the source table.
- Refresh all out-of-date projections.

The following recommendations apply to staging tables:
• To facilitate the backup process, create a unique schema for the staging table of each archiving operation.

• Specify new names for staging tables. This ensures that they do not contain partitions from previous move operations.

If the table does not exist, Vertica creates a table from the source table's definition, by calling `CREATE TABLE` with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table.

• Use staging names that enable other users to easily identify partition contents. For example, if a table is partitioned by dates, use a name that specifies a date or date range.

In the following example, `MOVE_PARTITIONS_TO_TABLE` specifies to move a single partition to the staging table `partn_backup.trades_200801`.

```
=> SELECT MOVE_PARTITIONS_TO_TABLE (  
  'prod_trades',  
  '200801',  
  '200801',  
  'partn_backup.trades_200801');
MOVE_PARTITIONS_TO_TABLE
------------------------------------------
1 distinct partition values moved at epoch 15.
(1 row)
```

Back Up the Staging Table

After you create a staging table, you archive it through an object-level backup using a `vbr` configuration file. For detailed information, see Backing Up and Restoring the Database.

Important: Vertica recommends performing a full database backup before the object-level backup, as a precaution against data loss. You can only restore object-level backups to the original database.

Drop the Staging Tables

After the backup is complete, you can drop the staging table as described in Dropping Tables.
Restoring Archived Partitions

You can restore partitions that you previously moved to an intermediate table, archived as an object-level backup, and then dropped.

Note: Restoring an archived partition requires that the original table definition is unchanged since the partition was archived and dropped. If the table definition changed, you can restore an archived partition with INSERT...SELECT statements, which are not described here.

These are the steps to restoring archived partitions:

1. Restore the backup of the intermediate table you saved when you moved one or more partitions to archive (see Archiving Partitions).
2. Move the restored partitions from the intermediate table to the original table.
3. Drop the intermediate table.

Swapping Partitions

SWAP_PARTITIONS_BETWEEN_TABLES combines the operations of DROP_PARTITIONS and MOVE_PARTITIONS_TO_TABLE as a single transaction. SWAP_PARTITIONS_BETWEEN_TABLES is useful if you regularly load partitioned data from one table into another and need to refresh partitions in the second table.

For example, you might have a table of revenue that is partitioned by date, and you routinely move data into it from a staging table. Occasionally, the staging table contains data for dates that are already in the target table. In this case, you must first remove partitions from the target table for those dates, then replace them with the corresponding partitions from the staging table. You can accomplish both tasks with a single call to SWAP_PARTITIONS_BETWEEN_TABLES.

By wrapping the drop and move operations within a single transaction, SWAP_PARTITIONS_BETWEEN_TABLES maintains integrity of the swapped data. If any task in the swap operation fails, the entire operation fails and is rolled back.
Example

The following example creates two partitioned tables and then swaps certain partitions between them.

Both tables have the same definition and have partitions for various year values. You swap the partitions where year = 2008 and year = 2009. Both tables have at least two rows to swap.

1. Create the customer_info table:

```sql
=> CREATE TABLE customer_info (  
customer_id INT NOT NULL,
first_name VARCHAR(25),
last_name VARCHAR(35),
city VARCHAR(25),
year INT NOT NULL)
ORDER BY last_name
PARTITION BY year;
```

2. Insert data into the customer_info table:

COPY customer_info FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|Joe|Smith|Denver|2008
>> 2|Bob|Jones|Boston|2008
>> 3|Silke|Muller|Frankfurt|2007
>> 4|Simone|Bernard|Paris|2014
>> 5|Vijay|Kumar|New Delhi|2010
>> 

3. View the table data:

```sql
=> SELECT * FROM customer_info ORDER BY year DESC;
customer_id | first_name | last_name | city      | year  
-----------------------------------------------------------------------
4 | Simone | Bernard | Paris | 2014  
5 | Vijay | Kumar | New Delhi | 2010  
1 | Joe | Smith | Denver | 2008  
2 | Bob | Jones | Boston | 2008  
3 | Silke | Muller | Frankfurt | 2007  
(5 rows)
```

4. Create a second table, member_info, that has the same definition as customer_info:

```sql
=> CREATE TABLE member_info LIKE customer_info INCLUDING PROJECTIONS;
CREATE TABLE
```
5. Insert data into the member_info table:

```sql
=> COPY member_info FROM STDIN;
Enter data to be copied followed by a newline. 
End with a backslash and a period on a line by itself. 
>> 1]Jane|Doe|Miami|2001 
>> 2]Mike|Brown|Chicago|2014 
>> 3]Patrick|OMalley|Dublin|2008 
>> \.
```

6. View the data in the member_info table:

```sql
=> SELECT * FROM member_info ORDER BY year DESC;
customer_id | first_name | last_name | city     | year
-------------|------------|-----------|----------|------
 2           | Mike       | Brown     | Chicago  | 2014 |
 4           | Ana        | Lopez     | Madrid   | 2009 |
 3           | Patrick    | OMalley   | Dublin   | 2008 |
 5           | Mike       | Green     | New York | 2008 |
 1           | Jane       | Doe       | Miami    | 2001 |

(5 rows)
```

7. To swap the partitions, run the SWAP_PARTITIONS_BETWEEN_TABLES function:

```sql
=> SELECT SWAP_PARTITIONS_BETWEEN_TABLES('customer_info', 2008, 2009, 'member_info');

SWAP_PARTITIONS_BETWEEN_TABLES

1 partition values from table customer_info and 2 partition values from table member_info are swapped at epoch 1045.

(1 row)
```

8. Query both tables to confirm that they swapped their respective 2008 and 2009 records:

```sql
=> SELECT * FROM customer_info ORDER BY year DESC;
customer_id | first_name | last_name | city     | year
-------------|------------|-----------|----------|------
 4           | Simone     | Bernard   | Paris    | 2014 |
 5           | Vijay      | Kumar     | New Delhi| 2010 |
 4           | Ana        | Lopez     | Madrid   | 2009 |
 3           | Patrick    | OMalley   | Dublin   | 2008 |
 5           | Mike       | Green     | New York | 2008 |
 3           | Silke      | Muller    | Frankfurt| 2007 |

(6 rows)
```

```sql
=> SELECT * FROM member_info ORDER BY year DESC;
customer_id | first_name | last_name | city     | year
-------------|------------|-----------|----------|------
 2           | Mike       | Brown     | Chicago  | 2014 |
 2           | Bob        | Jones     | Boston   | 2008 |
 1           | Joe        | Smith     | Denver   | 2008 |
```

Vertica Analytic Database (9.0.x)
Minimizing Partitions

By default, Vertica supports up to 1024 ROS containers to store partitions for a given projection (see Projection Parameters). A ROS container contains data that share the same partition key, or the same partition group key. Depending on the amount of data per partition, a partition or partition group can span multiple ROS containers.

Given this limit, it is inadvisable to partition a table on highly granular data—for example, on a TIMESTAMP column. Doing so can generate a very high number of partitions. If the number of partitions requires more than 1024 ROS containers, Vertica issues a ROS pushback warning and refuses to load more table data. A large number of ROS containers also can adversely affect DML operations such as DELETE, which requires Vertica to open all ROS containers.

In practice, it is unlikely you will approach this maximum. For optimal performance, Vertica recommends that the number of ungrouped partitions range between 10 and 20, and not exceed 50. This range is typically compatible with most business requirements.

You can also reduce the number of ROS containers by grouping partitions. For more information, see Partition Grouping and Hierarchical Partitioning.

Viewing Partition Storage Data

Vertica provides various ways to view how your table partitions are organized and stored:

- Query the PARTITIONS system table.
- Dump partition keys.

Querying PARTITIONS Table

The following table and projection definitions partition store_order data on order dates, and groups together partitions of the same year:

```sql
=> CREATE TABLE public.store_orders
   (order_no int, order_date timestamp NOT NULL, shipper varchar(20), ship_date date)
PARTITION BY ((order_date)::date) GROUP BY (date_trunc('year', (order_date)::date));
=> CREATE PROJECTION public.store_orders_unseg_super
```
AS SELECT order_no, order_date, shipper, ship_date FROM store_orders
ORDER BY order_no, order_date, shipper, ship_date UNSEGMENTED ALL NODES;

=> COPY store_orders FROM '/home/dbadmin/export_store_orders_data.txt' DIRECT;

After loading data into this table, you can query the PARTITIONS table to determine how many ROS containers store the grouped partitions for projection store_orders_unseg, across all nodes. Each node has four ROS containers, each container storing partitions of one partition group:

=> SELECT COUNT (partition_key) NumPartitions, ros_id, node_name FROM PARTITIONS
WHERE projection_name like 'store_orders_unseg%' GROUP BY ros_id, node_name ORDER BY node_name, NumPartitions;

<table>
<thead>
<tr>
<th>NumPartitions</th>
<th>ros_id</th>
<th>node_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>173</td>
<td>45035996275802477</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>211</td>
<td>45035996275802513</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>212</td>
<td>45035996275802489</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>213</td>
<td>45035996275802501</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>173</td>
<td>49539595901291955</td>
<td>v_vmart_node0002</td>
</tr>
<tr>
<td>211</td>
<td>49539595901291991</td>
<td>v_vmart_node0002</td>
</tr>
<tr>
<td>212</td>
<td>49539595901291967</td>
<td>v_vmart_node0002</td>
</tr>
<tr>
<td>213</td>
<td>49539595901291979</td>
<td>v_vmart_node0002</td>
</tr>
<tr>
<td>173</td>
<td>54043195528662659</td>
<td>v_vmart_node0003</td>
</tr>
<tr>
<td>211</td>
<td>54043195528662695</td>
<td>v_vmart_node0003</td>
</tr>
<tr>
<td>212</td>
<td>54043195528662671</td>
<td>v_vmart_node0003</td>
</tr>
<tr>
<td>213</td>
<td>54043195528662683</td>
<td>v_vmart_node0003</td>
</tr>
</tbody>
</table>

(12 rows)

### Dumping Partition Keys

Vertica provides several functions that let you inspect how individual partitions are stored on the cluster, at several levels:

- **DUMP_PARTITION_KEYS** dumps partition keys of all projections in the system.
- **DUMP_TABLE_PARTITION_KEYS** dumps partition keys of all projections for the specified table.
- **DUMP_PROJECTION_PARTITION_KEYS** dumps partition keys of the specified projection.

Given the previous table and projection, DUMP_PROJECTION_PARTITION_KEYS shows the contents of four ROS containers on each node:

=> SELECT DUMP_PROJECTION_PARTITION_KEYS('store_orders_unseg_super');

```sql
... Partition keys on node v_vmart_node0001
  Projection 'store_orders_unseg_super'
  Storage [ROS container]
```
### No of partition keys: 173


#### Storage [ROS container]
- No of partition keys: 212

#### Storage [ROS container]
- No of partition keys: 213

#### Storage [ROS container]
- No of partition keys: 211

#### Projection 'store_orders_unsegg_super'
- Storage [ROS container]
- No of partition keys: 173

---

### Active and Inactive Partitions

The Tuple Mover assumes that all loads and updates to a partitioned table are targeted to one or more partitions that it identifies as **active**. In general, the partitions with the largest partition keys—typically, the most recently created partitions—are regarded as active. As the partition ages, it commonly transitions to a mostly read-only workload and requires much less activity.

The configuration parameter **ActivePartitionCount** determines how many partitions are active for each partitioned table in the database. By default, **ActivePartitionCount** is set to 1, so the Tuple Mover assumes that only one partition—the newest one—is active. For
example, if a table is partitioned by month, the Tuple Mover expects that after the start of a new month, all activity on that table is directed at the new month's partition.

If loads and updates frequently occur to multiple partitions, you can adjust the ActivePartitionCount parameter accordingly. For example, if your database receives data for the current month as well as updates to the prior month, set ActivePartitionCount to 2. For tables partitioned by non-temporal attributes, set ActivePartitionCount to reflect the number of partitions that are loaded simultaneously.

The Tuple Mover uses the following algorithm to determine which partitions are older than others:

1. If one partition was created before the other partition, it is older.
2. If two partitions were created at the same time, but one partition was last updated earlier than the other partition, it is older.
3. If two partitions were created and last updated at the same time, the partition with the smaller key is considered older.

**Active Partition Groups**

If a table's partition clause includes a GROUP BY expression, Vertica applies ActivePartitionCount to the largest partition group key, and regards all the partitions in that group as active. If you group partitions with Vertica meta-function CALENDAR_HIERARCHY_DAY, the most recent date partitions are also grouped by day. Thus, the largest group key and largest partition group key are identical. In effect, this means that only the most recent partitions are active.

For more information about partition grouping, see Partition Grouping and Hierarchical Partitioning.

**Partition Pruning**

If a query predicate specifies a partitioning expression, the query optimizer evaluates the predicate against the ROS containers of the partitioned data. Each ROS container maintains the minimum and maximum values of its partition key data. The query optimizer uses this metadata to determine which ROS containers it needs to execute the query, and omits, or prunes, the remaining containers from the query plan. By minimizing the number of ROS containers that it must scan, the query optimizer enables faster execution of the query.

For example, a table might be partitioned by year as follows:
CREATE TABLE ... PARTITION BY EXTRACT(year FROM date);

Given this table definition, its projection data is partitioned into ROS containers according to year, one for each year—in this case, 2007, 2008, 2009.

The following query specifies the partition expression date:

=> SELECT ... WHERE date = '12-2-2009';

Given this query, the ROS containers that contain data for 2007 and 2008 fall outside the boundaries of the requested year (2009). The query optimizer prunes these containers from the query plan before the query executes:

<table>
<thead>
<tr>
<th>date</th>
<th>amount</th>
<th>date</th>
<th>amount</th>
<th>date</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/11/09</td>
<td>6</td>
<td>03/13/08</td>
<td>96</td>
<td>07/12/07</td>
<td>43</td>
</tr>
<tr>
<td>06/05/09</td>
<td>12</td>
<td>04/21/08</td>
<td>17</td>
<td>03/02/07</td>
<td>45</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12/02/09</td>
<td>8</td>
<td>12/02/08</td>
<td>7</td>
<td>12/02/07</td>
<td>68</td>
</tr>
<tr>
<td>Min: 01/01/09</td>
<td>Max: 12/31/09</td>
<td>Min: 01/01/08</td>
<td>Max: 12/31/08</td>
<td>Min: 01/01/07</td>
<td>Max: 12/31/07</td>
</tr>
</tbody>
</table>

Making Past Partitions Eligible for Pruning

The following procedure lets you make past partitions eligible for pruning. The easiest way to guarantee that all ROS containers are eligible is to:

1. Create a fact table with the same projections as the existing table.
2. Use INSERT..SELECT to populate the new table.
3. Drop the original table and rename the new table.

If the disk lacks enough space for a second copy of the fact table, follow this procedure:

1. Verify that the Tuple Mover finished all post-upgrade work—for example, when the following command shows no mergeout activity:

=> SELECT * FROM TUPLE_MOVER_OPERATIONS;

2. Identify which partitions need to be merged to get the ROS minimum/maximum values:
3. Insert a record into each partition that has ineligible ROS containers, and commit.

4. Delete each inserted record and commit again.

At this point, the Tuple Mover automatically merges ROS containers from past partitions.

### Verifying the ROS Merge

1. Query the TUPLE_MOVER_OPERATIONS table:

```sql
=> SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

2. Check for any partitions that need to be merged:

```sql
=> SELECT DISTINCT table_schema, projection_name, partition_key
    FROM partitions p LEFT OUTER JOIN vs_ros_min_max_values v
    ON p.ros_id = v.delid
    WHERE v.min_value IS null;
```

### Examples

Assume a table that is partitioned by time and will use queries that restrict data on time.

```sql
=> CREATE TABLE time ( tdate DATE NOT NULL, tnum INTEGER)
    PARTITION BY EXTRACT(year FROM tdate);
=> CREATE PROJECTION time_p (tdate, tnum) AS
    SELECT * FROM time ORDER BY tdate, tnum UNSEGMENTED ALL NODES;
```

**Note:** Projection sort order has no effect on partition pruning.

```sql
=> INSERT INTO time VALUES ('03/15/04' , 1);
=> INSERT INTO time VALUES ('03/15/05' , 2);
=> INSERT INTO time VALUES ('03/15/06' , 3);
=> INSERT INTO time VALUES ('03/15/06' , 4);
```

The data inserted in the previous series of commands are loaded into three ROS containers, one per year, as that is how the data is partitioned:
Here’s what happens when you query the `time` table:

- In this query, Vertica can omit container ROS2 because it is only looking for year 2004:

```sql
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004';
```

- In the next query, Vertica can omit two containers, ROS1 and ROS3:

```sql
=> SELECT COUNT(*) FROM time WHERE tdate = '10/07/2005';
```

- The following query has an additional predicate on the `tnum` column for which no minimum/maximum values are maintained. In addition, the use of logical operator OR is not supported, so no ROS elimination occurs:

```sql
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004' OR tnum = 7;
```
About Constraints

*Constraints* specify rules on what values can go into a column. Examples of constraints:

- Primary key
- Foreign key
- Unique
- Check
- Not NULL

Using constraints can help you maintain data integrity in one or more columns. Do not define constraints on columns unless you expect to keep the data consistent.

Vertica can use constraints to perform optimizations (such as the optimized MERGE) that assume the data is consistent.
Adding Constraints

Add constraints on one or more table columns using the following SQL commands:

- **CREATE TABLE**: Add a constraint on one or more columns.
- **ALTER TABLE**: Add or drop a constraint on one or more columns.

Vertica recommends naming a constraint but it is optional; if you specify the CONSTRAINT keyword, you must give a name for the constraint.

The examples that follow illustrate several ways of adding constraints. For additional details, see:

- Primary Key Constraints
- Foreign Key Constraints
- Unique Constraints
- Check constraints
- Not NULL Constraints

Adding Column Constraints with CREATE TABLE

There are several ways to add a constraint on a column using CREATE TABLE:

- On the column definition using the CONSTRAINT keyword, which requires that you assign a constraint name, in this example, dim1PK:

```sql
CREATE TABLE dim1 (  c1 INTEGER CONSTRAINT dim1PK PRIMARY KEY,  c2 INTEGER  );
```

- On the column definition, omitting the CONSTRAINT keyword. When you omit the CONSTRAINT keyword, you cannot specify a constraint name:

```sql
CREATE TABLE dim1 (  c1 INTEGER PRIMARY KEY,  c2 INTEGER  );
```
After the column definition, using the CONSTRAINT keyword and assigning a name, in this example, dim1PK:

```sql
CREATE TABLE dim1 (  c1 INTEGER,  c2 INTEGER,  CONSTRAINT dim1pk PRIMARY KEY(c1)) ;
```

After the column definition, omitting the CONSTRAINT keyword:

```sql
CREATE TABLE dim1 (  c1 INTEGER,  c2 INTEGER,  PRIMARY KEY(c1)) ;
```

### Adding Two Constraints on a Column

To add more than one constraint on a column, specify the constraints one after another when you create the table column. For example, the following statement enforces both not NULL and unique constraints on the customer_key column, indicating that the column values cannot be NULL and must be unique:

```sql
CREATE TABLE test1 (  id INTEGER NOT NULL UNIQUE,  ... ) ;
```

### Adding a Foreign Key Constraint on a Column

There are four ways to add a foreign key constraint on a column using CREATE TABLE. The FOREIGN KEY keywords are not valid on the column definition, only after the column definition:

- On the column definition, use the CONSTRAINT and REFERENCES keywords and name the constraint, in this example, fact1dim1PK. This example creates a column with a named foreign key constraint referencing the table (dim1) with the primary key (c1):

  ```sql
  CREATE TABLE Fact1 (  c1 INTEGER CONSTRAINT fact1dim1FK REFERENCES dim1(c1),  c2 INTEGER  ) ;
  ```
On the column definition, omit the CONSTRAINT keyword and use the REFERENCES keyword with the table name and column:

```sql
CREATE TABLE Fact1 ( c1 INTEGER REFERENCES dim1(c1),
                     c2 INTEGER)
```

After the column definition, use the CONSTRAINT, FOREIGN KEY, and REFERENCES keywords and name the constraint:

```sql
CREATE TABLE Fact1 ( c1 INTEGER,
                     c2 INTEGER,
                     CONSTRAINT fk1 FOREIGN KEY(c1) REFERENCES dim1(c1)
)
```

After the column definition, omitting the CONSTRAINT keyword:

```sql
CREATE TABLE Fact1 ( c1 INTEGER,
                     c2 INTEGER,
                     FOREIGN KEY(c1) REFERENCES dim1(c1)
)
```

Each of the following ALTER TABLE statements adds a foreign key constraint on an existing column, with and without using the CONSTRAINT keyword:

```sql
ALTER TABLE Fact2
ADD CONSTRAINT fk1 FOREIGN KEY (c1) REFERENCES dim2(c1);
```

or

```sql
ALTER TABLE Fact2
ADD FOREIGN KEY (c1) REFERENCES dim2(c1);
```

For additional details, see Foreign Key Constraints.

### Adding Multicolumn Constraints

The following example defines a primary key constraint on multiple columns by first defining the table columns (c1 and c2), and then specifying both columns in a PRIMARY KEY clause:

```sql
CREATE TABLE Dim ( c1 INTEGER,
                   c2 INTEGER,
                   PRIMARY KEY (c1, c2)
)
```
To specify multicolumn (compound) primary keys, the following example uses CREATE TABLE to define the columns. After creating the table, ALTER TABLE defines the compound primary key and names it dim2PK:

```sql
CREATE TABLE dim2 ( c1 INTEGER, c2 INTEGER, c3 INTEGER NOT NULL, c4 INTEGER UNIQUE )
ALTER TABLE dim2
  ADD CONSTRAINT dim2PK PRIMARY KEY (c1, c2);
```

In the next example, you define a compound primary key as part of the CREATE TABLE statement. Then you specify the matching foreign key constraint to table dim2 using CREATE TABLE and ALTER TABLE:

```sql
CREATE TABLE dim2 ( c1 INTEGER, c2 INTEGER, c3 INTEGER NOT NULL, c4 INTEGER UNIQUE, PRIMARY KEY (c1, c2) )
CREATE TABLE fact2 ( c1 INTEGER, c2 INTEGER, c3 INTEGER NOT NULL, c4 INTEGER UNIQUE )
ALTER TABLE fact2
  ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2(c1, c2);
```

Specify a foreign key constraint using a reference to the table that contains the primary key. In the ADD CONSTRAINT clause, the REFERENCES column names are optional. The following ALTER TABLE statement is equivalent to the previous ALTER TABLE statement:

```sql
ALTER TABLE fact2 ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2;
```

## Adding Constraints on Tables with Existing Data

When you add a constraint on a column with existing data, Vertica:

- Verifies the validity of the column values only if you are adding a primary key, unique key, or check constraint enabled for automatic enforcement.

- Does not verify the validity of column values for other constraint types.

If your data does not conform to the declared non-enabled constraints, your queries could yield unexpected results.
Use `ANALYZE_CONSTRAINTS` to check for constraint violations in your column. If you find violations, use the `ALTER COLUMN SET/DROP` parameters of the `ALTER TABLE` statement to apply or remove a constraint on an existing column.

Note: You can configure your system to automatically enforce primary key, unique key and check constraints during DML. For information on automatic enforcement, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

### Altering Column Constraints

The following example uses `ALTER TABLE` to add column b with not NULL and default 5 constraints to a table `test6`:

```sql
CREATE TABLE test6 (a INT);
ALTER TABLE test6 ADD COLUMN b INT DEFAULT 5 NOT NULL;
```

Use `ALTER TABLE` with the `ALTER COLUMN` and `SET NOT NULL` clauses to add the constraint on column a in table `test6`:

```sql
ALTER TABLE test6 ALTER COLUMN a SET NOT NULL;
```

Use the `SET NOT NULL` or `DROP NOT NULL` clause to add or remove a not NULL column constraint:

```sql
=> ALTER TABLE T1 ALTER COLUMN x SET NOT NULL;
=> ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

Use these clauses so that the column has the proper constraints when you have added or removed a primary key constraint on a column. You can also use them any time you want to add or remove the NOT NULL constraint.

Note: A PRIMARY KEY constraint includes a NOT NULL constraint. However, if you drop the PRIMARY KEY constraint on a column, the NOT NULL constraint remains on that column.

### Enforcing Constraints

Check constraints are enforced by default unless you disable individual constraints when you create or alter the constraint, or set the parameter `EnableNewCheckConstraintsByDefault` to 0 (disabled). See check constraints for more information.
To maximize query performance, Vertica checks for primary key and foreign key violations when loading into the fact table of a pre-join projection. For more details, see Enforcing Primary Key and Foreign Key Constraints.

Vertica checks for not NULL constraint violations when loading data, but it does not check for unique constraint violations for constraints that are not enabled.

To validate table data on constraints that are not enabled, load data without committing it by using the COPY with the NO COMMIT option. Then perform a post-load check using the ANALYZE_CONSTRAINTS function. If constraint violations are found, you can roll back the load because you have not committed it. For more details, see Detecting Constraint Violations with ANALYZE_CONSTRAINTS.

**Note:** Vertica enforces check constraints automatically by default. You can also enforce primary key and unique key constraints automatically. See Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

### See Also

- ALTER TABLE
- CREATE TABLE
- COPY
- ANALYZE_CONSTRAINTS

### Primary Key Constraints

A primary key (PK) is a single column or combination of columns (called a *compound key*) that uniquely identifies each row in a table. A primary key constraint contains unique, non-null values.

When you apply the primary key constraint, the NOT NULL and unique constraints are added implicitly. You do not need to specify them when you create the column. However, if you remove the primary key constraint, the NOT NULL constraint continues to apply to the column. To remove the NOT NULL constraint after removing the primary key constraint, use the ALTER COLUMN DROP NOT NULL parameter of the ALTER TABLE statement (see Dropping Constraints).

The following example shows how you can add a primary key constraint on the employee_id field:
CREATE TABLE employees (employee_id INTEGER PRIMARY KEY);

Alternatively, you can add a primary key constraint after the column is created:

```
CREATE TABLE employees (employee_id INTEGER);
ALTER TABLE employees
  ADD PRIMARY KEY (employee_id);
```

**Note:** If you specify a primary key constraint using ALTER TABLE, the system returns the following message, which is informational only. The primary key constraint is added to the designated column.

```
WARNING 2623: Column "employee_id" definition changed to NOT NULL
```

You can also use primary keys to constrain more than one column:

```
CREATE TABLE employees (employee_id INTEGER, 
  employee_gender CHAR(1),
  PRIMARY KEY (employee_id, employee_gender)
);
```

When you enable automatic enforcement of primary or unique key constraints, Vertica applies enforcement for:

- INSERT
- UPDATE
- MERGE
- COPY
- COPY_PARTITIONS_TO_TABLE
- MOVE_PARTITIONS_TO_TABLE
- SWAP_PARTITIONS_BETWEEN_TABLES

Alternatively, rather than automatic enforcement, you can use ANALYZE_CONSTRAINTS to validate primary and unique key constraints after issuing these statements. For more information on enabling and disabling primary key constraints, refer to [Enforcing Primary Key, Unique Key, and Check Constraints Automatically](#).
Foreign Key Constraints

A foreign key (FK) is a column that is used to join a table to other tables to ensure referential integrity of the data. A foreign key constraint requires that a column contain only values from the primary key column on a specific dimension table.

You can create a foreign key constraint in the CREATE TABLE statement, or you can define a foreign key constraint using ALTER TABLE.

A column with a foreign key constraint can contain NULL values if it does not also have a not NULL constraint, even though the NULL value does not appear in the dimension table’s primary key column. This allows rows to be inserted into the table even if the foreign key is not yet known.

You can add a foreign key constraint by referencing the table that contains the primary key. The columns in the referenced table do not need to be specified explicitly.

Examples

Create a table called inventory to store inventory data:

```sql
CREATE TABLE inventory (  
date_key INTEGER NOT NULL,  
product_key INTEGER NOT NULL,  
warehouse_key INTEGER NOT NULL,  
...  
);  
```

Create a table called warehouse to store warehouse information:

```sql
CREATE TABLE warehouse (  
warehouse_key INTEGER NOT NULL PRIMARY KEY,  
warehouse_name VARCHAR(20),  
...  
);  
```

To ensure referential integrity between the inventory and warehouse tables, define a foreign key constraint called fk_inventory_warehouse on the inventory table that references the warehouse table:

```sql
ALTER TABLE inventory  
ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key) REFERENCES warehouse(warehouse_key);  
```

In this example, the inventory table is the referencing table and the warehouse table is the referenced table.
You can also create the foreign key constraint in the CREATE TABLE statement that creates the inventory table, eliminating the need for the ALTER TABLE statement. If you do not specify one or more columns, the PRIMARY KEY of the referenced table is used:

```
CREATE TABLE inventory (
    date_key INTEGER NOT NULL,
    product_key INTEGER NOT NULL,
    warehouse_key INTEGER NOT NULL REFERENCES warehouse(warehouse_key),
    ...;
```

A foreign key can also constrain and reference multiple columns. The following example uses CREATE TABLE to add a foreign key constraint to a pair of columns:

```
CREATE TABLE t1 (  c1 INTEGER PRIMARY KEY,  
c2 INTEGER,  
c3 INTEGER,  
FOREIGN KEY (c2, c3) REFERENCES other_table (c1, c2)  
);
```

The following two examples use ALTER TABLE to add a foreign key constraint to a pair of columns. When you use the CONSTRAINT keyword, you must specify a constraint name:

```
ALTER TABLE t ADD FOREIGN KEY (a, b) REFERENCES other_table(c, d);  
ALTER TABLE t ADD CONSTRAINT fk_cname FOREIGN KEY (a, b) REFERENCES other_table(c, d);
```

Note: The FOREIGN KEY keywords are valid only after the column definition, not on the column definition.

### Unique Constraints

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all rows in the table.

### How to Verify Unique Constraints

Vertica allows you to add a (non-enabled) unique constraint to a column. You can then insert data into that column, regardless of whether that constraint is not unique with respect to other values in that column. If your data does not conform to the declared non-enabled constraints, your queries could yield unexpected results.

You can use `ANALYZE_CONSTRAINTS` to check for constraint violations, or you can enable automatic enforcement of unique key constraints. For more information on enabling and disabling unique key constraints, refer to Enforcing Primary Key, Unique Key, and Check Constraints Automatically.
Add Unique Column Constraints

There are several ways to add a unique constraint on a column. If you use the CONSTRAINT keyword, you must specify a constraint name. The following example adds a UNIQUE constraint on the product_key column and names it product_key_UK:

```
CREATE TABLE product (product_key INTEGER NOT NULL CONSTRAINT product_key_UK UNIQUE,
...);
```

Vertica recommends naming constraints, but it is optional:

```
CREATE TABLE product (product_key INTEGER NOT NULL UNIQUE,
...);
```

You can specify the constraint after the column definition, with and without naming it:

```
CREATE TABLE product (product_key INTEGER NOT NULL,
...,
CONSTRAINT product_key_UK UNIQUE (product_key)
);
CREATE TABLE product (product_key INTEGER NOT NULL,
...,
UNIQUE (product_key)
);
```

You can also use ALTER TABLE to specify a unique constraint. This example names the constraint product_key_UK:

```
ALTER TABLE product ADD CONSTRAINT product_key_UK UNIQUE (product_key);
```

You can use CREATE TABLE and ALTER TABLE to specify unique constraints on multiple columns. If a unique constraint refers to a group of columns, separate the column names using commas. The column listing specifies that the combination of values in the indicated columns is unique across the whole table, though any one of the columns need not be (and ordinarily isn't) unique:

```
CREATE TABLE dim1 (c1 INTEGER,
    c2 INTEGER,
    c3 INTEGER,
    UNIQUE (c1, c2)
);
```
Check Constraints

A check constraint specifies a SQL predicate (Boolean expression) that is evaluated independently on each row of a table. The predicate cannot access data stored in other tables or database objects, such as sequences. It also cannot access data in other rows of the current table.

As with other constraints, check constraints help ensure data integrity. By default, Vertica automatically enforces check constraints. You have the option of turning this feature off for specific constraints when you create or alter them. You can also set the configuration parameter EnableNewCheckConstraintsByDefault to 0 (disabled), which automatically disables all check constraints that you subsequently create or alter. You can override this parameter by explicitly enabling a specific constraint.

For more information on enabling and disabling constraints, refer to Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

The following examples show how you can use check constraints:

Add a constraint named smplcheck to the column n1 without specifically enabling or disabling the constraint:

```
=> CREATE TABLE checksample (n1 int CONSTRAINT smplcheck CHECK(n1<50),n2 int, n3 int);
```

Add a constraint in the disabled state:

```
=> CREATE TABLE checksample2 (n1 int CONSTRAINT smplcheck2 CHECK(n1<50) DISABLED,n2 int, n3 int);
```

Use ALTER TABLE to add a constraint to an existing table:

```
=> ALTER TABLE checksample ADD CONSTRAINT smplcheck2 CHECK(n2<100);
```

Use ALTER TABLE to disable an existing constraint:

```
=> ALTER TABLE checksample ALTER CONSTRAINT smplcheck2 DISABLED;
```

When you enable automatic enforcement of check constraints, Vertica applies enforcement for:

- INSERT
- UPDATE
- MERGE
- COPY
- COPY_PARTITIONS_TO_TABLE (Vertica enforces check constraints on the target table.)
- MOVE_PARTITIONS_TO_TABLE (Vertica enforces check constraints on the target table.)
- SWAP_PARTITIONS_BETWEEN_TABLES (Vertica enforces check constraints on both tables.)

Alternatively, rather than automatic enforcement, you can use ANALYZE CONSTRAINTS to validate constraints after issuing these statements. ANALYZE CONSTRAINTS reports all constraint violations. For more information on enabling and disabling check constraints, refer to Enforcing Primary Key, Unique Key, and Check Constraints Automatically

### Allowed Syntax for Check Constraint Predicates

A check constraint predicate can include:

- Arithmetic and concatenated string operators
- Logical operators — Such as AND, OR, NOT
- WHERE prepositions — Such as CASE, IN, LIKE, BETWEEN, IS (NOT) NULL
- Calls to certain types of functions — Immutable SQL macros, immutable build-in SQL functions (for example, length()) and user-defined scalar functions that are marked as immutable in the component factory.

The following examples show you how you can define hypothetical check constraints on a table. The examples assume that the table contains the referenced attributes (column names), defined as appropriate types. (The last one is an unnamed check constraint.)

```sql
CONSTRAINT chk_pos_quant CHECK (quantity > 0)
CONSTRAINT chk_pqe CHECK (price*quantity = extended_price)
CONSTRAINT size_sml CHECK (size in ('small', 'medium', 'large', 'x-large'))
CHECK (regexp_like(dept_name, '^[a-zA-Z]+$', 'i') OR (dept_name = 'inside sales'))
```

### Restrictions on Check Constraints

A check constraint expression must evaluate to a Boolean value. However, Vertica does not support implicit conversion to a Boolean value. For example, Vertica would return an error for the following (invalid) check constraint:
CHECK (1) -- produces an error
CHECK ('hello') -- produces an error

You must always enclose a check constraint in parentheses:

```
check (quantity > 0)
```

A check constraint expression cannot include any of the following elements:

- Subqueries — CHECK (dept_id in (SELECT id FROM dept))
- Aggregates — CHECK (quantity < sum(quantity)/2)
- Window functions — CHECK (RANK() over () < 3)
- SQL meta-functions — CHECK (START_REFRESH('') = 0)
- References to the epoch column
- References to other tables or objects (for example, sequences), or system context
- Invocation of functions that are not immutable in time and space

### Check Constraints and Nulls

If a check constraint expression evaluates to an unknown for a given row because a column within the expression contains a null, the row passes the constraint condition. Vertica evaluates the predicate and considers it satisfied if it resolves to either true or unknown. For example, check (quantity > 0) passes validation if quantity is null. This result differs from how a WHERE clause would work. With a WHERE clause, the row would not be included in the result set.

You can prohibit nulls in a check constraint by explicitly including a null check in the check constraint expression. For example, CHECK (quantity IS NOT NULL AND (quantity > 0)). You could alternatively include a not null constraint separate from the check constraint.

### Check Constraints and SQL Macros

A check constraint can call a SQL macro (a function written in SQL) if the macro is immutable. An *immutable macro* always returns the same value for a given set of arguments.

When you include a macro, Vertica determines if it is immutable. If it is not, Vertica rolls back the DDL statement.
This example shows you can create a macro name, `mycompute`, and then use it within a check constraint expression:

```sql
=> CREATE OR REPLACE FUNCTION mycompute(j int, name1 varchar)
RETURN int AS BEGIN RETURN (j + length(name1)); END;
=> ALTER TABLE sampletable
ADD CONSTRAINT chk_compute
CHECK(mycompute(weekly_hours, name1))<50);
```

**Check Constraints and User-Defined Extensions (UDxs)**

A check constraint can call user-defined scalar function (UDSFs), but not other kinds of UDxs. To use a UDSF, you must first mark it as immutable in the UDx factory.

If you use a UDSF within a check constraint, you must verify that the immutable tag on the referenced functions is correct and that the constraint handles null values properly. Otherwise, the check constraint may not work as you intended. In addition, Vertica evaluates the predicate of an enabled check constraint on every row that is loaded or updated. For performance reasons, you may want to avoid invoking a computationally expensive check constraint.

To view a sample UDSF that you can use within a check constraint, refer to [C++ Example: Calling a UDSF from a Check Constraint](#).

**Dropping a Check Constraint**

Use `ALTER TABLE` with the `DROP CONSTRAINT` option to explicitly drop a constraint.

If you drop a table the includes check constraints, Vertica automatically deletes all check constraints associated with the table. The `CASCADE` option is not needed.

If you drop a table column (or function object):

- If you do not specify the `CASCADE` option, Vertica returns an error if any check constraint references the column. Vertica returns the error even if the constraint is disabled.
- If you specify `CASCADE`, Vertica drops check constraints (both enabled and disabled) along with the table column (or function object).

If you rename a table column, Vertica updates the constraint predicate to use the new name.
Not NULL Constraints

A not NULL constraint specifies that a column cannot contain a null value. This means that new rows cannot be inserted or updated unless you specify a value for this column.

You can apply the not NULL constraint when you create a column in a new table, and when you add a column to an existing table (ALTER TABLE . . ADD COLUMN). You can also add or drop the not NULL constraint on an existing column:

- ALTER TABLE t ALTER COLUMN x SET NOT NULL
- ALTER TABLE t ALTER COLUMN x DROP NOT NULL

Important: Using the [SET | DROP] NOT NULL clause does not validate whether column data conforms to the NOT NULL constraint. Use ANALYZE_CONSTRAINTS to check for constraint violations in a table.

The not NULL constraint is implicitly applied to a column when you add the PRIMARY KEY (PK) constraint. When you designate a column as a primary key, you do not need to specify the not NULL constraint.

However, if you remove the primary key constraint, the not NULL constraint still applies to the column. Use the ALTER COLUMN . . DROP NOT NULL clause of the ALTER TABLE statement to drop the not NULL constraint after dropping the primary key constraint.

The following statement enforces a not NULL constraint on the customer_key column, specifying that the column cannot accept NULL values.

```
CREATE TABLE customer (customer_key INTEGER NOT NULL,
  ...);
```
Dropping Constraints

To drop named constraints, use the `ALTER TABLE` command.

The following example drops the constraint `fact2fk`:

```sql
=> ALTER TABLE fact2 DROP CONSTRAINT fact2fk;
```

To drop constraints that you did not assign a name to, query the system table `TABLE_CONSTRAINTS`, which returns both system-generated and user-named constraint names. For example:

```sql
=> SELECT * FROM TABLE_CONSTRAINTS;
```

If you do not specify a constraint name, Vertica assigns a constraint name that is unique to that table. In the following output, note the system-generated constraint name `C_PRIMARY` and the user-defined constraint name `fk_inventory_date`:

```
- [ RECORD 1 ]--------------------------------
constraint_id | 45035996273707984
constraint_name | C_PRIMARY
constraint_schema_id | 45035996273704966
constraint_key_count | 1
foreign_key_count | 0
table_id | 45035996273707982
foreign_table_id | 0
constraint_type | p
- [ ... ]--------------------------------
- [ RECORD 9 ]--------------------------------
constraint_id | 45035996273708016
constraint_name | fk_inventory_date
constraint_schema_id | 0
constraint_key_count | 1
foreign_key_count | 1
table_id | 45035996273708014
foreign_table_id | 45035996273707994
constraint_type | f
```

Once you know the name of the constraint, you can then drop it using the `ALTER TABLE` command. (If you do not know the table name, use `table_id` to retrieve `table_name` from the `ALL_TABLES` table.)

Remove NOT NULL Constraints

When a column is a primary key and you drop the primary key constraint, the column retains the NOT NULL constraint. To specify that the column now can contain NULL values, use `DROP
NOT NULL] to remove the NOT NULL constraint.

Remove (Drop) a NOT NULL constraint on the column using [DROP NOT NULL]:

```sql
ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

Important: Using the [SET | DROP] NOT NULL clause does not validate whether the column data conforms to the NOT NULL constraint. Use ANALYZE_CONSTRAINTS to check for constraint violations in a table.

**Limitations of Dropping Constraints**

- You cannot drop a primary key constraint if another table has a foreign key constraint that references the primary key.

- If you drop a primary or foreign key constraint, the system does not automatically drop the not NULL constraint on a column. You need to manually drop this constraint if you no longer want it.

- If you drop an enabled primary or unique key constraint, the system drops the associated projection if one was automatically created.

**See Also**

ALTER TABLE
Enforcing Primary Key and Foreign Key Constraints

Enforcing (Non-Enabled) Primary Key Constraints

Unless you enable enforcement of primary key constraints, Vertica does not enforce the uniqueness of primary key values when they are loaded into a table. Thus, a key enforcement error can occur unless one dimension row uniquely matches each foreign key value when the table is joined to a dimension table during a query:

Note: Consider using sequences or auto-incrementing columns for primary key columns, which guarantees uniqueness and avoids the constraint enforcement problem and associated overhead. For more information, see Using Sequences.

For information on automatic enforcement of primary key constraints during DML, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

Foreign Key Constraint Violations

A table’s foreign key constraints are not enforced during data load. Thus, it is possible to load data that causes a constraint violation. Subsequently, a constraint violation error can occur when:

- An inner join query is processed.
- An outer join is treated as an inner join due to the presence of foreign key.

Detecting Constraint Violations Before You Commit Data

To detect constraint violations, you can load data without committing it using the COPY statement with the NO COMMIT option, and then perform a post-load check using the ANALYZECONSTRAINTS function. If constraint violations exist, you can roll back the load because you have not committed it. For more details, see Detecting Constraint Violations with ANALYZECONSTRAINTS.
You can also configure your system to automatically enforce primary key, unique key, and check constraints during DML. For information on automatic enforcement, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

**Enforcing Primary Key, Unique Key, and Check Constraints Automatically**

When you create a new constraint with `CREATE TABLE` or `ALTER TABLE`, you can specify whether the constraint will be automatically enforced. You can also alter a constraint with `ALTER TABLE` (using the `ALTER CONSTRAINT` parameter) and specify whether it will be automatically enforced. You enable or disable individual constraints specifically using the `ENABLED` or `DISABLED` options.

In addition, you can create multi-column constraints with `CREATE TABLE` or `ALTER TABLE`. All primary key and unique key constraints are defined at the table level. Check constraints are defined at the column or table level.

By checking any system table with an `is_enabled` column, you can confirm whether a primary key, unique key, or check constraint is currently enabled. The system tables that include an `is_enabled` column are `CONSTRAINT_COLUMNS`, `TABLE_CONSTRAINTS`, and `PRIMARY_KEYS`.

Automatic enforcement applies to current table content and content you later add to the table.

- **Enabling a Constraint on an Empty Table** — If you create an enabled constraint on an empty table, the constraint is enforced on any content you later add to that table.

- **Enabling a Constraint on a Populated Table** — If you use `ALTER TABLE` to either enable an existing constraint or add a new constraint that is enabled, the constraint is immediately enforced for the current content, and is enforced for content you subsequently add to the table.

  **Important:** If validation of the current content fails, Vertica completely rolls back the `ALTER TABLE` DDL statement that caused the failure.

If you do not specify the `ENABLED` or `DISABLED` option when you create a constraint, the system relies on the setting of the configuration parameter for the respective constraint:

- **EnableNewPrimaryKeysByDefault** — If you specifically create a new primary key constraint but do not enable or disable it, the system relies on the value of the parameter
EnableNewPrimaryKeysByDefault. If the parameter is set to 1 (enabled), the constraint you created is automatically enforced even though you did not specifically enable it when you created it.

- **EnableNewUniqueKeysByDefault** — If you specifically create a new unique key constraint but do not enable or disable it, the system relies on the value of the parameter EnableNewUniqueKeysByDefault.

- **EnableNewCheckConstraintsByDefault** — If you specifically create a new check constraint but do not enable or disable it, the system relies on the value of the parameter EnableNewCheckConstraintsByDefault. This parameter is the only one of the three that is set to 1 (enabled) by default.

In regards to constraint enablement, you can:

- **Enable or Disable a Constraint When Creating** — You can specifically enable or disable when you create the constraint using CREATE TABLE or ALTER TABLE. If you do so, the constraint remains enabled or disabled regardless of the setting of the parameters EnableNewPrimaryKeysByDefault, EnableNewUniqueKeysByDefault, or EnableNewCheckConstraintsByDefault.

- **Create a Constraint Without Enabling or Disabling** — You can also create a constraint using CREATE TABLE or add a constraint using ALTER TABLE without specifically enabling or disabling it using the ENABLED or DISABLED keyword. If you do so, Vertica looks at the setting of the parameters at the moment you create or alter the constraint to determine whether that constraint is enabled or disabled. Note that, if you alter an existing constraint using ALTER TABLE ALTER CONSTRAINT, you must specifically enable or disable it using either the ENABLED or DISABLED keyword.

**Important:** When creating a constraint without enabling it, Vertica uses the settings of EnableNewPrimaryKeysByDefault, EnableNewUniqueKeysByDefault, and EnableNewCheckConstraintsByDefault that are in effect at the time of creation.

The following figure summarizes constraint enablement.
Enabling or Disabling Automatic Enforcement of Individual Constraints

To enable or disable individual constraints, use the `CREATE TABLE` or `ALTER TABLE` statement with the ENABLED or DISABLED options, as shown in the following examples.

The following sample uses `ALTER TABLE` to create and enable a primary key constraint on a sample table called `mytable`.

```
ALTER TABLE mytable ADD CONSTRAINT primarysample PRIMARY KEY(id) ENABLED;
```

The following sample specifically disables the constraint.
ALTER TABLE mytable ALTER CONSTRAINT primarysample DISABLED;

The following sample uses CREATE TABLE to create a primary key constraint without explicitly enabling it. In this case, the constraint is enabled only if EnableNewPrimaryKeysByDefault is also enabled. If EnableNewPrimaryKeysByDefault is set to 1 (enabled), then this constraint is enforced. If EnableNewPrimaryKeysByDefault is at its default setting (disabled), then this constraint is not enforced.

CREATE TABLE mytable (id INT PRIMARY KEY);

The following sample uses CREATE TABLE to create a primary key constraint and enable it. This statement enables the constraint regardless of how you set the parameter EnableNewPrimaryKeysByDefault.

CREATE TABLE mytable (id INT PRIMARY KEY ENABLED);

The following example uses CREATE TABLE to create and specifically disable the check constraint, chyear. This statement disables the constraint regardless of the setting of how you set the parameter EnableNewCheckConstraintsByDefault.

CREATE TABLE emphire (hire_date INT, termination_date INT CONSTRAINT chyear CHECK (termination_date >= hire_date) DISABLED, firstnameof VARCHAR, lastnameof VARCHAR);

Checking Whether Constraints Are Enabled

Use the SELECT statement to list constraints and confirm whether they are enabled or disabled.

This example shows a query that lists all tables along with their associated primary key, unique, and check constraint types. The query also indicates whether the constraints are enabled or disabled.

select table_name, constraint_name, constraint_type, is_enabled from v_catalog.constraint_columns where constraint_type in ('p', 'u', 'c') order by table_name;

The following output shows the results of this query. The constraint_type column indicates whether the constraint is a primary key, unique key, or check constraint (p, u, or c, respectively). The is_enabled column indicates whether the constraint is enabled or disabled (t or f respectively).

<table>
<thead>
<tr>
<th>table_name</th>
<th>constraint_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>table01</td>
<td>pksample</td>
<td>p</td>
<td>t</td>
</tr>
</tbody>
</table>
The following example is similar but shows how you can create a query that lists associated columns instead of tables. You could add both tables and columns to the same query, if you want.

```
select column_name, constraint_name, constraint_type, is_enabled
from v_catalog.constraint_columns where constraint_type in ('p', 'u', 'c') order by column_name;
```

Sample output follows.

<table>
<thead>
<tr>
<th>column_name</th>
<th>constraint_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col1_key</td>
<td>pksample</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>vendor_key</td>
<td>uniquesample</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>zip_key</td>
<td>checksample</td>
<td>c</td>
<td>t</td>
</tr>
</tbody>
</table>

(2 rows)

The following example statement shows how to create a sample table with a multi-column constraint.

```
CREATE TABLE table09 (column1 int, column2 int, CONSTRAINT multicsample PRIMARY KEY (column1, column2) ENABLED);
```

Here's the output listing associated columns.

<table>
<thead>
<tr>
<th>column_name</th>
<th>constraint_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>column1</td>
<td>multicsample</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>column2</td>
<td>multicsample</td>
<td>p</td>
<td>t</td>
</tr>
</tbody>
</table>

(2 rows)

Choosing Default Enforcement for Newly Declared or Modified Constraints

The EnableNewPrimaryKeysByDefault and EnableNewUniqueKeysByDefault parameter settings govern automatic enforcement of primary key and unique key constraints. The parameter
EnableNewCheckConstraintsByDefault governs automatic enforcement of check constraints, and is the only one of the three parameters that is enabled by default.

Important: If you disable enforcement with a parameter, the constraints you create or modify are not enforced unless you specifically enable them using `CREATE TABLE` or `ALTER TABLE`.

You do not need to restart your database once you have set these parameters.

- To enable or disable enforcement of newly created primary keys, set the parameter `EnableNewPrimaryKeysByDefault`. To disable, keep the default setting of 0. To enforce the constraints, set `EnableNewPrimaryKeysByDefault` to 1 to enable.

  ```sql
  ALTER DATABASE VMart SET EnableNewPrimaryKeysByDefault = 1;
  ```

- To enable or disable enforcement of newly created constraints for unique keys, set the parameter `EnableNewUniqueKeysByDefault`. To disable, keep the default setting of 0. Set `EnableNewUniqueKeysByDefault` to 1 to enable.

  ```sql
  ALTER DATABASE VMart SET EnableNewUniqueKeysByDefault = 1;
  ```

- To enable or disable enforcement of newly created check constraints, set the parameter `EnableNewCheckConstraintsByDefault`. To enable, keep the default setting of 1. Set `EnableNewCheckConstraintsByDefault` to 0 to disable.

  ```sql
  ALTER DATABASE VMart SET EnableNewCheckConstraintsByDefault = 0;
  ```

You can check individual parameter values using `SHOW CURRENT`:

```sql
=> show current Enablenewcheckconstraintsbydefault;

<table>
<thead>
<tr>
<th>level</th>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE</td>
<td>EnableNewCheckConstraintsByDefault</td>
<td>0</td>
</tr>
</tbody>
</table>
```

When you upgrade to Vertica 9.0.x, the primary and unique key constraints in any tables you carry over are disabled. Existing constraints are not automatically enforced. To enable existing constraints and make them automatically enforceable, manually enable each constraint using the `ALTER TABLE ALTER CONSTRAINT` statement. This statement triggers constraint enforcement for the existing table contents. Statements roll back if one or more violations occur.
How Enabled Primary and Unique Key Constraints Affect Locks

If you enable automatic constraint enforcement, Vertica uses an Insert-Validate (IV) lock. The IV lock is needed for operations where the system performs constraint validation for enabled PRIMARY or UNIQUE key constraints. Such operations can include INSERT, COPY, MERGE, UPDATE, MOVE_PARTITION_TO_TABLE.

How DML Operates with Constraints

With enforced primary or unique key constraints, DML operates in two-stages.

- **First Stage.** The first stage is the same as it would be for an unenforced constraint, taking, for example, an I lock. (This could be done by several sessions concurrently loading the same table.)

- **Second Stage.** The second stage includes the IV lock. Vertica takes an IV lock to make sure that the data does not violate your constraint. After performing this check, Vertica can commit the data.

Delays in Bulk Loading Caused by Constraint Validation

In bulk load situations, some transactions could be temporarily blocked while primary or unique key constraints are validated. For example:

You could have three sessions (for example, sessions 1, 2 and 3). Each session concurrently has an I lock for a bulk load. Session 1 takes an IV lock to validate constraints. Only one session can hold an IV lock on a given table; other sessions can continue loading the table while holding I locks.

Sessions 2 and 3 wait for session 1 to validate constraints, and then commit, releasing the IV lock. (If session 1 fails, the statement rolls back, and the next session can obtain the IV lock. While sessions can load the table in parallel, an IV lock requires that sessions takes turns obtaining the IV lock for the final stage of constraint validation.)

For information on lock modes and compatibility and conversion matrices, see Lock Modes in Vertica Concepts. See also the LOCKS and LOCK_USAGE sections in the SQL Reference Manual.
Projections for Enabled Primary and Unique Key Constraints

To enforce primary and unique key constraints, Vertica creates special key projections as needed in response to DML or DDL, which are checked for constraint violations. If a constraint violation occurs, Vertica rolls back the statement and any special key projection it created. The system returns an error specifying the unique or primary key constraint that was violated.

If you have added a constraint on a table that is empty, Vertica does not immediately create a special key projection for that constraint. Vertica defers creation of a special key projection until the first row of data is added to the table using a DML or COPY statement. If you add a constraint to a populated table, Vertica chooses an existing projection for enforcement of the constraint, if possible. If none of the existing projections are sufficient to validate the constraint, Vertica creates a new projection for the enabled constraint.

You can check primary and unique key constraint projections by querying the `PROJECTIONS` systems table under the `V_CATALOG` Schema. Each entry applying to a key constraint projection include the column name `IS_KEY_CONSTRAINT_PROJECTION`.

If you drop an enabled primary or unique key constraint, the system may drop an associated projection if one was automatically created. You can drop a specific projection even if a key constraint is enabled:

- If you drop a specific projection without including the `CASCADE` option in your `DROP` statement, Vertica issues a warning about dropping a projection for an enabled constraint.

- If you drop a specific projection and include the `CASCADE` option in your `DROP` statement, Vertica drops the projection without issuing the warning.

In either case, the next time Vertica needs to enforce the constraint for DML, the system creates a new special key projection, unless an existing projection can enforce the same enabled constraint. The time it takes to regenerate a key projection depends upon the volume of the table.

Note: If you subsequently use `ANALYZE_CONSTRAINTS` on a table that has enabled primary or unique key constraints (and thus their associated projections), `ANALYZECONSTRAINTS` can leverage the projections previously created for enforcement, resulting in a performance improvement for `ANALYZE_CONSTRAINTS`.

Deciding Whether to Enable Primary Key, Unique Key, and Check Constraints

You have the option to choose automatic enforcement of primary key, unique key, and check constraints. Depending upon your specific scenario, you can either enable this feature, or use ANALYZE_CONSTRAINTS to validate constraints. Consider these factors:

- **Benefits of Enabling Primary Key, Unique Key, and Check Constraints**
- **Considerations Before Enabling Constraints**
- **Where Constraints Are Enforced**
- **Impact of Floating Point Values in Primary Keys When Using Automatic Enforcement**
- **Constraint Enforcement Limitations**

For more information on using ANALYZE_CONSTRAINTS and how automatic enforcement differs, see the Administrator's Guide section, Detecting Constraint Violations with ANALYZE_CONSTRAINTS.

**Benefits of Enabling Primary Key, Unique Key, and Check Constraints**

When you enable primary key, unique key, or check constraints, Vertica validates data before it is inserted. Because you do not need to validate data using ANALYZE_CONSTRAINTS after insertion, query speed improves.

Having enabled key constraints, particularly on primary keys, can help the optimizer produce faster query plans, particularly for joins. When a table has an enabled primary key constraint, the optimizer can assume that it has no rows with duplicate values across the key set.

Vertica automatically creates special purpose projections, if necessary, to enforce enabled key constraints. In some cases Vertica can use an existing projection instead.

**Considerations Before Enabling Constraints**

Multiple factors affect performance. The enforcement process can slow DML and bulk loading.
If you are doing bulk loads, consider the size of your tables and the number of columns in your keys. You could decide to disable automatic enforcement for fact tables, which tend to be larger, but enable enforcement for dimension tables. For fact tables, you could choose manual constraint validation using `ANALYZE_CONSTRAINTS`, and avoid the load-time overhead of automatic validation.

When you enable automatic enforcement of primary key, unique key, or check constraints, statement rollbacks occur if validation fails during DML. Vertica completely rolls back the statement causing the failure. When deciding to enable automatic enforcement of constraints, consider the impact of statements rolling back on violations. For example, you issue ten insert statements, none of which have committed. If the sixth statement introduces a duplicate, that statement is rolled back. The other statements that do not introduce duplicates can commit.

**Note:** Vertica performs primary key, unique key, and check constraint enforcement at the SQL statement level rather than the transaction level. You cannot defer enforcement until transaction commit.

### Where Constraints Are Enforced

Automatic enforcement of constraints occurs in:

- **INSERT statements** — Both in single row insertions, and in an INSERT statement that includes the SELECT parameter.

- **Bulk loads** — On bulk loads that use the COPY statement.

- **UPDATE or MERGE statements** — All UPDATE and MERGE statements.

- **Meta functions** — On COPY_PARTITIONS_TO_TABLE, MOVE_PARTITIONS_TO_TABLE and SWAP_PARTITIONS_BETWEEN_TABLES.

- **ALTER TABLE statements** — On statements that include either the ADD CONSTRAINT or ALTER CONSTRAINT parameters where you are enabling a constraint and the table has existing data.

### Impact of Floating Point Values in Primary Keys When Using Automatic Enforcement

Vertica allows NaN, +Inf, and -Inf values in a FLOAT type column, even if the column is part of a primary key. Because FLOAT types provide imprecise arithmetic, Vertica recommends that you not use columns with floating point values within primary keys.
If you do decide to use a FLOAT type within a primary key, note the following in regards to primary key enforcement. (This behavior is the same regardless of whether you enable an automatic constraint or check constraints manually with \texttt{ANALYZE\_CONSTRAINTS}.)

- For the purpose of enforcing key constraints, Vertica considers two NaNs, (or two +Inf, or two –Inf) values to be equal.

- If a table has an enabled single column primary key constraint of type FLOAT, only one tuple can have a NaN value for the column. Otherwise, the constraint is violated. This is also true for +Inf and –Inf values. Note that this differs from the IEEE 754 standard, which specifies that multiple NaN values are different from each other.

- A join on a single column that contains FLOAT values fails if the table that includes a primary key contains multiple tuples with two NaNs (or +Inf, or –Inf) values.

For information on floating point type, see \texttt{DOUBLE PRECISION (FLOAT)}.

Constraint Enforcement Limitations

You can only enable or disable automatic enforcement for primary key, unique key, and check constraints. Vertica does not support automatic enforcement of foreign keys and referential integrity. You can manually validate foreign key constraints using the meta-function \texttt{ANALYZE\_CONSTRAINTS}.

Vertica does not support automatic enforcement of constraints on external tables.

Limitations on Using Automatic Enforcement for Local and Global Temporary Tables

This section includes limitations and related notes on using automatic enforcement of primary and unique key constraints with local and global temporary tables. For general information on temporary tables, see \texttt{Creating Temporary Tables}.

Limitations for Local and Global Temporary Tables

Vertica displays an error message if you add an enabled constraint to a local or global temporary table that contains data. Vertica displays the error because it cannot create
projected for enabled constraints on a temporary table if that table is already populated with data.

**Limitations Specific to Global Temporary Tables**

You cannot use ALTER TABLE to add a new or enable an existing primary or unique key constraint on a global temporary table. Use CREATE TABLE to enable a constraint on a global temporary table.

You can use ALTER TABLE to add a new or enable an existing primary or unique key constraint on a local temporary table if the local temporary table is empty.

**Note:** You can use ALTER TABLE to disable an already enabled primary or unique key constraint on a global temporary table.

**Reporting Constraint Violations**

For enabled constraints, Vertica reports multiple constraint violations before roll back. It reports multiple violations for primary key and unique constraints in all circumstances (for example, when you are loading or moving data). However, Vertica behaves differently for check constraints.

**Reporting Primary Key and Unique Constraint Violations**

The following example shows multiple duplicates in a file named `sampledatafile.tbl`. The sample table named `pktest` includes an enabled primary key comprised of the column `cost`. Vertica reports the duplicates when you attempt to load the table.

```
VMart=> CREATE TABLE pktest(cost int constraint costcons primary key enabled,destination int,timemorn int,finish int);
CREATE TABLE

VMart=> copy pktest from '/home/dbadmin/test/sampledatafile.tbl' delimiter ',';

ERROR 6745: Duplicate key values: 'cost=106'
-- violates constraint 'public.pktest.costcons'

DETAIL: Additional violations:
Constraint 'public.pktest.costcons':
duplicate key values: 'cost=110'; 'cost=114'; 'cost=115'; 'cost=116'; 'cost=117'; 'cost=118'; 'cost=119'; 'cost=120'; 'cost=121'; 'cost=122'; 'cost=123'; 'cost=124'; 'cost=125'; 'cost=126'; 'cost=136'; 'cost=137'; 'cost=138'; 'cost=139'; 'cost=150'; 'cost=151'; 'cost=20'; 'cost=200'; 'cost=251'; 'cost=252'; 'cost=255'; 'cost=257'; 'cost=258'; 'cost=261'; 'cost=263';
```


As shown in the example, Vertica provides more complete information on the first violation, including the name of the constraint that was violated:

```sql
ERROR 6745: Duplicate key values: 'cost=106'
  -- violates constraint 'public.pktest.costcons'
```

For subsequent violations, it reports abbreviated detail, up to a maximum of 30 violations. If there are more than 30 violations, Vertica adds the note "there were additional errors" at the end of the list of violations:

```sql
DETAIL: Additional violations:
Constraint 'public.pktest.costcons':
duplicate key values: 'cost=110'; 'cost=114'; 'cost=115'; 'cost=116'; 'cost=117';
'cost=118'; 'cost=119'; 'cost=120'; 'cost=121'; 'cost=122'; 'cost=123'; 'cost=124';
'cost=125'; 'cost=135'; 'cost=136'; 'cost=137'; 'cost=138'; 'cost=139'; 'cost=150';
'cost=151'; 'cost=20'; 'cost=200'; 'cost=201'; 'cost=251'; 'cost=252'; 'cost=255'; 'cost=257';
'cost=258'; 'cost=261'; 'cost=263';
```

Refer to the section Primary Key Constraints for general information on where Vertica applies enforcement for enabled primary key and unique constraints.

## Reporting Multiple Check Constraints

For enabled check constraints, Vertica reports only one constraint violation before roll back with three exceptions:

- You add or change an enabled check constraint on a table that already contains data.
- You move or copy partitions to a table, or when you swap partitions between tables.
- You copy a table using the Vertica function COPY_TABLE.

Refer to the section Check Constraints for general information on where Vertica applies enforcement for enabled check constraints.

This example shows multiple check constraint violations using the Vertica function COPY_TABLE.

```sql
VMart=> create table a(a int);
CREATE TABLE
VMart=> create table b(a int check(a>5));
CREATE TABLE
VMart=>
```
Naming Constraints

You can create and name constraints in different ways, as the following examples show. Vertica recommends that you name your constraints. If you do not name your constraints, Vertica assigns a name. The examples show various statement syntaxes for creating and altering named and unnamed constraints.

Naming a Primary Key Constraint

Create a table with a named primary key, where the named constraint is myconstraint.

```sql
=> CREATE TABLE addapk (col1 INT CONSTRAINT myconstraint PRIMARY KEY ENABLED, col2 INT);
CREATE TABLE
```

Query the PRIMARY_KEYS table to view the constraint.

```sql
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');
column_name | constraint_type | is_enabled
-------------+----------------+---------
```

Vertica Documentation
Administrator's Guide

Vertica Analytic Database (9.0.x)
Creating a Primary Key Constraint Without Naming

Create a table with a primary key constraint, where you do not name the constraint. In this case, Vertica assigns a name to the constraint.

```sql
=> CREATE TABLE addapk (col1 INT PRIMARY KEY ENABLED, col2 INT);
```

Query the PRIMARY_KEYS table to view the constraint. Vertica has named the constraint C_PRIMARY.

```sql
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');
```

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_PRIMARY</td>
<td>col1</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Altering Named and Unnamed Constraints

Alter a table that includes a constraint that you named.

```sql
=> ALTER TABLE addapk ALTER CONSTRAINT myconstraint DISABLED;
```

Alter a table with a constraint to which Vertica has assigned a name.

```sql
=> ALTER TABLE addapk ALTER CONSTRAINT C_PRIMARY DISABLED;
```

Alter a table by adding a newly named constraint.

```sql
=> CREATE TABLE addapk (col1 INT, col2 INT);
CREATE TABLE

=> ALTER TABLE addapk ADD CONSTRAINT myconstraint PRIMARY KEY (COL1) ENABLED;
WARNING 2623: Column "col1" definition changed to NOT NULL
ALTER TABLE
```

Alter a table, and let Vertica name the constraint.

```sql
=> CREATE TABLE addapk (col1 INT, col2 INT);
CREATE TABLE

=> ALTER TABLE addapk ADD PRIMARY KEY (COL1) ENABLED;
WARNING 2623: Column "col1" definition changed to NOT NULL
Creating a Multi-Column Constraint

Create a table with a primary key that comprises three columns.

=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT, PRIMARY KEY (col1,col2,col3) ENABLED);

Query the PRIMARY_KEYS table to view the multi-column primary key constraint.

=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_PRIMARY</td>
<td>col1</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>C_PRIMARY</td>
<td>col2</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>C_PRIMARY</td>
<td>col3</td>
<td>p</td>
<td>t</td>
</tr>
</tbody>
</table>

(3 rows)

Create a table with a primary key that comprises three columns, and name the constraint.

=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT, CONSTRAINT myconstraint PRIMARY KEY (col1,col2,col3) ENABLED);

Add a three-column primary key after you create the table.

=> CREATE TABLE addapk (col1 INT, col2 INT, col3 INT, col4 INT, col5 INT);
CREATE TABLE
=> ALTER TABLE addapk ADD CONSTRAINT myconstraint PRIMARY KEY (col1,col2,col3) ENABLED;
WARNING 2623: Column "col1" definition changed to NOT NULL
WARNING 2623: Column "col2" definition changed to NOT NULL
WARNING 2623: Column "col3" definition changed to NOT NULL
ALTER TABLE

Creating Multiple Constraint Types

The following example adds three constraints, including primary key, unique, and check constraints.

=> CREATE TABLE addapk (col1 INT CONSTRAINT myPKconstraint PRIMARY KEY ENABLED, col12 INT CONSTRAINT myUconstraint UNIQUE ENABLED, col13 INT CONSTRAINT myCHconstraint CHECK(col1<col2));
CREATE TABLE

Query the CONSTRAINT_COLUMNS table to view the constraints.
=> SELECT column_name, constraint_name, constraint_type, is_enabled FROM constraint_columns WHERE constraint_type IN ('c','p','u') AND table_name IN ('addapk');

<table>
<thead>
<tr>
<th>column_name</th>
<th>constraint_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col1</td>
<td>myCHconstraint</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>col1</td>
<td>myPKconstraint</td>
<td>p</td>
<td>t</td>
</tr>
<tr>
<td>col2</td>
<td>myUconstraint</td>
<td>u</td>
<td>t</td>
</tr>
<tr>
<td>col2</td>
<td>myCHconstraint</td>
<td>c</td>
<td>t</td>
</tr>
</tbody>
</table>

(4 rows)
Detecting Constraint Violations with ANALYZE_CONSTRAINTS

Use the ANALYZE_CONSTRAINTS function to manually validate table constraints.

Ways to Use ANALYZE_CONSTRAINTS

You can use ANALYZE_CONSTRAINTS instead of (or as a supplement to) automatic enforcement of primary key, unique key, and check constraints. For information on automatic enforcement, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

If you do enable primary key, unique key, and check constraints, note that ANALYZE_CONSTRAINTS does not check whether the constraints are disabled or enabled. You can use ANALYZE_CONSTRAINTS where:

- Primary key, unique key, and check constraints are disabled.
- Enabled and disabled constraints are mixed.

You can use ANALYZE_CONSTRAINTS to validate referential integrity of foreign keys. Vertica does not support automatic enforcement of foreign keys.

How to Use ANALYZE_CONSTRAINTS to Detect Violations

The ANALYZE_CONSTRAINTS function analyzes and reports on constraint violations within the current schema search path. To check for constraint violations:

- Pass an empty argument to check for violations on all tables within the current schema.
- Pass a single table argument to check for violations on the specified table.
- Pass two arguments, a table name and a column or list of columns, to check for violations in those columns.

See the examples in ANALYZE_CONSTRAINTS for more information.
How ANALYZE_CONSTRAINTS Differs from Automatic Constraint Enforcement

Use ANALYZE_CONSTRAINTS as a reporting mechanism to find constraint violations. ANALYZE_CONSTRAINTS differs from automatic enforcement in that it does not enforce constraints.

Automatic enforcement of primary key, unique key, and check constraints does enforce constraints.

If you have set your constraints for automatic enforcement, during a bulk load Vertica reports on one row and rolls back the statement that introduced the constraint violation. If you want to see a report for all constraint violations:

- Disable the automatic enforcement of your contraints.

- After the bulk load, run ANALYZE_CONSTRAINTS. ANALYZE_CONSTRAINTS analyzes and reports all rows that violate a constraint, without removing the offending rows.

Impact of Floating Point Values In Primary Keys When Using ANALYZECONSTRAINTS

Vertica allows NaN, +Inf, and -Inf values in a FLOAT type column, even if the column is part of a primary key. Because FLOAT types provide imprecise arithmetic, Vertica recommends that you not use columns with floating point values within primary keys.

If you do decide to use a FLOAT type within a primary key, note the following in regards to primary key enforcement. (This behavior is the same regardless of whether you enable an automatic constraint or check constraints manually with ANALYZE_CONSTRAINTS.)

- For the purpose of enforcing key constraints, Vertica considers two NaNs, (or two +Inf, or two –Inf) values to be equal.

- If a table has an enabled single column primary key constraint of type FLOAT, only one tuple can have a NaN value for the column. Otherwise, the constraint is violated. This is also true for +Inf and –Inf values. Note that this differs from the IEEE 754 standard, which specifies that multiple NaN values are different from each other.

- A join on a single column that contains FLOAT values fails if the table that includes a primary key contains multiple tuples with two NaNs (or +Inf, or –Inf) values.

For information on floating point type, see DOUBLE PRECISION (FLOAT).
Examples

If you provide the following inputs, Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice.

```sql
=> CREATE TABLE t1(c1 INT);
=> ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
=> CREATE PROJECTION t1_p (c1) AS SELECT *
    FROM t1 UNSEGMENTED ALL NODES;
=> INSERT INTO t1 values (10);
=> INSERT INTO t1 values (10); -- Duplicate primary key value
=> \x
Expanded display is on.
=> SELECT ANALYZE_CONSTRAINTS('t1');
```

- [ RECORD 1 ]----------------
  | Schema Name | public |
  | Table Name  | t1     |
  | Column Names| c1     |
  | Constraint Name | pk_t1 |
  | Constraint Type | PRIMARY |
  | Column Values | ('10') |

In the preceding query, if you give the second INSERT statement any different value, the result is 0 rows (no violations).

In the following example, create a table that contains three integer columns, one a unique key and one a primary key.

```sql
=> CREATE TABLE table_1(
    a INTEGER,
    b_UK INTEGER UNIQUE,
    c_PK INTEGER PRIMARY KEY
);
```

Insert some values into table table_1 and commit the changes:

```sql
=> INSERT INTO table_1 values (1, 1, 1);
=> COMMIT;
```

Run ANALYZE_CONSTRAINTS on table table_1. No constraint violations are reported:

```sql
=> SELECT ANALYZE_CONSTRAINTS('table_1');
(No rows)
```

Insert duplicate unique and primary key values and run ANALYZE_CONSTRAINTS on table table_1 again. Vertica returns two violations: one against the primary key and one against the unique key:
=> INSERT INTO table_1 VALUES (1, 1, 1);
=> COMMIT;
=> SELECT ANALYZE_CONSTRAINTS('table_1');

The following example shows how you can look for constraint violations on only the unique key in the table `table_1`, qualified with its schema name.

=> SELECT ANALYZE_CONSTRAINTS('public.table_1', 'b_UK');

The following example shows that you can specify the same column more than once; ANALYZE_CONSTRAINTS, however, returns the violation only once.

=> SELECT ANALYZE_CONSTRAINTS('table_1', 'c_PK, C_PK');

The following example creates a new table, `table_2`, and inserts a foreign key and different (character) data types.

=> CREATE TABLE table_2 (
  x VARCHAR(3),
  y_PK VARCHAR(4),
  z_FK INTEGER REFERENCES table_1(c_PK));
Alter the table to create a multicoloumn unique key and multicoloumn foreign key and create superprojections:

```sql
=> ALTER TABLE table_2
    ADD CONSTRAINT table_2_multiuk PRIMARY KEY (x, y_PK);
WARNING 2623: Column "x" definition changed to NOT NULL
WARNING 2623: Column "y_PK" definition changed to NOT NULL
```

The following statement inserts a missing foreign key (0) into table `table_2` and commits the changes.

```sql
=> INSERT INTO table_2 VALUES ('r1', 'Xpk1', 0);
=> COMMIT;
```

Checking for constraints on the table `table_2` in the `public` schema detects a foreign key violation:

```sql
=> SELECT ANALYZE_CONSTRAINTS('public.table_2');
```

```
| Schema Name | public |
| Table Name  | table_2 |
| Column Names | z_FK |
| Constraint Name | C_FOREIGN |
| Constraint Type | FOREIGN |
| Column Values | ('0') |
```

Now add a duplicate value into the unique key and commit the changes:

```sql
=> INSERT INTO table_2 VALUES ('r2', 'Xpk1', 1);
=> INSERT INTO table_2 VALUES ('r1', 'Xpk1', 1);
=> COMMIT;
```

Checking for constraint violations on table `table_2` detects the duplicate unique key error:

```sql
=> SELECT ANALYZE_CONSTRAINTS('table_2');
```

```
<p>| Schema Name | public |
| Table Name  | table_2 |
| Column Names | z_FK |
| Constraint Name | C_FOREIGN |
| Constraint Type | FOREIGN |</p>
<table>
<thead>
<tr>
<th>Column Values</th>
<th>('0')</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Name</td>
<td>public</td>
</tr>
<tr>
<td>Table Name</td>
<td>table_2</td>
</tr>
<tr>
<td>Column Names</td>
<td>x, y_PK</td>
</tr>
<tr>
<td>Constraint Name</td>
<td>table_2_multiuk</td>
</tr>
<tr>
<td>Constraint Type</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>Column Values</td>
<td>('r1', 'Xpk1')</td>
</tr>
</tbody>
</table>
```

Create a table with multicoloumn foreign key and then create the superprojections:
\( \Rightarrow \) CREATE TABLE table_3(
    z_fk1 VARCHAR(3),
    z_fk2 VARCHAR(4));
\( \Rightarrow \) ALTER TABLE table_3
    ADD CONSTRAINT table_3_multifk FOREIGN KEY (z_fk1, z_fk2)
    REFERENCES table_2(x, y_PK);

Insert a foreign key that matches a foreign key in table table_2 and commit the changes:

\( \Rightarrow \) INSERT INTO table_3 VALUES ('r1', 'Xpk1');
\( \Rightarrow \) COMMIT;

Check for constraints on table table_3. The query detects no violations:

\( \Rightarrow \) SELECT ANALYZE_CONSTRAINTS('table_3');
(No rows)

Add a value that does not match and commit the change:

\( \Rightarrow \) INSERT INTO table_3 VALUES ('r1', 'NONE');
\( \Rightarrow \) COMMIT;

Check for constraints on table table_3. The query detects a foreign key violation:

\( \Rightarrow \) SELECT ANALYZE_CONSTRAINTS('table_3');

- [ RECORD 1 ]-------------------
  Schema Name | public
  Table Name   | table_3
  Column Names | z_fk1, z_fk2
  Constraint Name | table_3_multifk
  Constraint Type | FOREIGN
  Column Values | ('r1', 'NONE')

To analyze all constraints on all tables, issue the following statement.

SELECT ANALYZE_CONSTRAINTS('');

Clean up your database:

\( \Rightarrow \) DROP TABLE table_1 CASCADE;
\( \Rightarrow \) DROP TABLE table_2 CASCADE;
\( \Rightarrow \) DROP TABLE table_3 CASCADE;

To learn how to remove violating rows, see the DISABLE_DUPLICATE_KEY_ERROR function.
Fixing Constraint Violations

When Vertica finds duplicate primary key or unique values at run time, use the DISABLE_DUPLICATE_KEY_ERROR function to suppress error messaging. Queries execute as though no constraints are defined on the schema and the effects are session scoped.

The DISABLE_DUPLICATE_KEY_ERROR function is for use only for key constraints that are not automatically enabled.

Caution: When called, DISABLE_DUPLICATE_KEY_ERROR suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a pre-join projection. Correct the violations and reenable integrity checking with REENABLE_DUPLICATE_KEY_ERROR.

Reenabling Error Reporting

If you ran DISABLE_DUPLICATE_KEY_ERROR to suppress error reporting while fixing duplicate key violations, you can get incorrect query results going forward. As soon as you fix the violations, run the REENABLE_DUPLICATE_KEY_ERROR function to restore the default behavior of error reporting.

The effects of this function are session scoped.
Managing Queries

This section covers the following topics:

- **Query Plans**: Describes how Vertica creates and uses query plans, which optimize access to information in the Vertica database.

- **Directed Queries**: Shows how to save query plan information.
Query Plans

A query plan is a sequence of step-like paths that the Vertica cost-based query optimizer uses to execute queries. Vertica can produce different query plans for a given query. For each query plan, the query optimizer evaluates the data to be queried: number of rows, column statistics such as number of distinct values (cardinality), distribution of data across nodes. It also evaluates available resources such as CPUs and network topology, and other environment factors. The query optimizer uses this information to develop several potential plans. It then compares plans and chooses one, generally the plan with the lowest cost.

Viewing Query Plans

You can obtain query plans in two ways:

- The EXPLAIN statement outputs query plans in various text formats.

- Management Console provides a graphical interface for viewing query plans. For detailed information, see Managing Queries in MC in Using Management Console.

You can also observe the real-time flow of data through a query plan by querying the system table QUERY_PLAN_PROFILES. For more information, see Profiling Query Plans.
EXPLAIN-Generated Query Plans

**EXPLAIN** returns the optimizer's query plan for executing a specified query. For example:

```sql
EXPLAIN SELECT customer_name, customer_state FROM customer_dimension WHERE customer_state IN ('MA', 'NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;
```

**QUERY PLAN DESCRIPTION:**
```
- Access Path:
  | +--SELECT LIMIT 10 [Cost: 365, Rows: 10] (PATH ID: 0)
  |   | Output Only: 10 tuples
  |   | Execute on: Query Initiator
  |   | ++---- SORT [TOPK] [Cost: 365, Rows: 544] (PATH ID: 1)
  |   |   | Order: customer_dimension.customer_name ASC
  |   |   | Output Only: 10 tuples
  |   |   | Execute on: Query Initiator
  |   | ++---- STORAGE ACCESS for customer_dimension [Cost: 326, Rows: 544] (PATH ID: 2)
  |   |   | Projection: public.customer_dimension.DBD_1_rep_VMartDesign_node0001
  |   |   | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
  |   |   | Filter: (customer_dimension.customer_gender = 'Male')
  |   |   | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
  |   |   | Execute on: Query Initiator
```

You can use **EXPLAIN** to evaluate choices that the optimizer makes with respect to a given query. If you think query performance is less than optimal, run it through the Database Designer. For more information, see [Incremental Design](#) and [Reducing Run-time of Queries](#).

**Query Plan Cost Estimation**

The query optimizer chooses a query plan based on cost estimates. The query optimizer uses information from a number of sources to develop potential plans and determine their relative costs. These include:

- **Number of table rows**
- **Column statistics**, including: number of distinct values (cardinality), minimum/maximum values, distribution of values, and disk space usage
- **Access path** that is likely to require fewest I/O operations, and lowest CPU, memory, and network usage
- **Available eligible projections**
- Join options: join types (merge versus hash joins), join order
- Query predicates
- Data segmentation across cluster nodes

Many important optimizer decisions rely on statistics, which the query optimizer uses to determine the final plan to execute a query. Therefore, it is important that statistics be up to date. Without reasonably accurate statistics, the optimizer could choose a suboptimal plan, which might affect query performance.

Vertica provides hints about statistics in the query plan. See Query Plan Statistics.

Cost versus Execution Runtime

Although costs correlate to query runtime, they do not provide an estimate of actual runtime. For example, if the optimizer determines that Plan A costs twice as much as Plan B, it is likely that Plan A will require more time to run. However, this cost estimate does not necessarily indicate that Plan A will run twice as long as Plan B.

Also, plan costs for different queries are not directly comparable. For example, if the estimated cost of Plan X for query1 is greater than the cost of Plan Y for query2, it is not necessarily true that Plan X's runtime is greater than Plan Y's runtime.

EXPLAIN Output Options

By default, EXPLAIN output represents the query plan as a hierarchy, where each level, or path, represents a single database operation that the optimizer uses to execute a query. EXPLAIN output also appends DOT language source so you can display this output graphically with open source Graphviz tools.

You can control the generated output by qualifying EXPLAIN with the following options:

- **JSON** produces the query plan in JSON format.
- **VERBOSE** increases the amount of detail in the rendered query plan. This option is valid for default and JSON output.
- **LOCAL** (on a multi-node database) shows the local query plans assigned to each node, which together comprise the total (global) query plan. If you omit this option, Vertica shows
only the global query plan. Local query plan are shown only in DOT language source, which
can be rendered in Graphviz.

- ANNOTATED: Embeds optimizer hints that encapsulate the query plan for this query. Vertica
uses these hints to create directed queries. For an example of usage, see Using Optimizer-
Generated and Custom Directed Queries Together.

**JSON Output**

The following EXPLAIN statement specifies to produce output in JSON format:

```sql
=> EXPLAIN JSON SELECT customer_name, customer_state FROM customer_dimension
   WHERE customer_state IN ('MA', 'NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;
```

**QUERY PLAN DESCRIPTION:**

```
Opt Vertica Options
------------------
PLAN_OUTPUT_JSON

EXPLAIN JSON SELECT customer_name, customer_state FROM customer_dimension
   WHERE customer_state IN ('MA', 'NH') AND customer_gender='Male' ORDER BY customer_name LIMIT 10;
```

**JSON format:**

```
{
   "PATH_ID" : 0,
   "PATH_NAME" : "SELECT",
   "EXTRA" : " LIMIT 10",
   "COST" : 365.000000,
   "ROWS" : 10.000000,
   "COST_STATUS" : "GOOD",
   "TUPLE_LIMIT" : 10,
   "EXECUTE_NODE" : "Query Initiator",
   "INPUT" : {
      "PATH_ID" : 1,
      "PATH_NAME" : "SORT",
      "EXTRA" : "[TOPK]",
      "COST" : 365.000000,
      "ROWS" : 544.000000,
      "COST_STATUS" : "GOOD",
      "ORDER" : ["customer_dimension.customer_name", "customer_dimension.customer_state"],
      "TUPLE_LIMIT" : 10,
      "EXECUTE_NODE" : "Query Initiator",
      "INPUT" : {
         "PATH_ID" : 2,
         "PATH_NAME" : "STORAGE ACCESS",
         "EXTRA" : "for customer_dimension",
         "COST" : 326.000000,
         "ROWS" : 544.000000,
```
VERBOOSE Output

The following EXPLAIN statement specifies the same query shown earlier, but this time specifies to produce verbose output (added information is set off in bold):

```
"COST_STATUS" : "GOOD",
"PROJECTION" : "public.customer_dimension_DBD_1_rep_VMartDesign_node0001",
"MATERIALIZE" : ["customer_dimension.customer_state", "customer_dimension.customer_name"],
"FILTER" : ["(customer_dimension.customer_gender = 'Male')", "(customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))"],
"EXECUTE_NODE" : "Query Initiator"
}
}
End JSON format

(51 rows)

QUERY PLAN DESCRIPTION:
---------------------

Opt Vertica Options
---------------------

PLAN_OUTPUT_SUPER_VERBOSE

EXPLAIN VERBOSE SELECT customer_name, customer_state FROM customer_dimension
WHERE customer_state IN ('MA', 'NH') AND customer_gender='Male'
ORDER BY customer_name LIMIT 10;

Access Path:
+--SELECT LIMIT 10 [Cost: 365.000000, Rows: 10.000000 Disk(B): 0.000000 CPU(B): 0.000000 Memory(B): 0.000000 Network(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274] (PATH ID: 0)
  | Output Only: 10 tuples
  | Execute on: Query Initiator
  | Sort Key: (customer_dimension.customer_name)
  | LDISTRIBUTE: UNSEGMENTED
  +---> SORT [TOPK] [Cost: 365.000000, Rows: 544.000000 Disk(B): 0.000000 CPU(B): 1275443.584695 Memory(B): 149856.000000 Network(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274] (PATH ID: 1)
    |     | Order: customer_dimension.customer_name ASC
    |     | Output Only: 10 tuples
    |     | Execute on: Query Initiator
    |     | Sort Key: (customer_dimension.customer_name)
    |     | LDISTRIBUTE: UNSEGMENTED
    +---> STORAGE ACCESS for customer_dimension [Cost: 326.000000, Rows: 544.000000 Disk(B): 0.000000 CPU(B): 0.000000 Memory(B): 0.000000 Network(B): 0.000000 Parallelism: 1.000000] [OutRowSz (B): 274] (PATH ID: 2)
      |     | Column Cost Aspects: [Disk(B): 410928.432432 CPU(B): 111588.324324 Memory(B): 4498570.572304 Network(B): 0.000000 Parallelism: 1.000000 ]
      |     | Projection: public.customer_dimension_DBD_1_rep_VMartDesign_node0001
      |     | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
      |     | Filter: (customer_dimension.customer_gender = 'Male')/* sel=0.351351 ndv= 2 */
      |     | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))/* sel=0.030928 ndv= 4 */
```
Local Output

The following statement includes the LOCAL option:

```sql
=> EXPLAIN LOCAL SELECT store_name, store_city, store_state
    FROM store.store_dimension ORDER BY store_state ASC, store_city ASC;
```

The output includes GraphViz source, which describes the local query plans assigned to each node. For example, output for this statement on a three-node database includes a GraphViz description of the following query plan for one node (v_vmart_node0003):

```graphviz
digraph G {
  graph [rankdir=BT, label= "v_vmart_node0003\n", labelloc=t, labeljust=l ordering=out]
  0[label = "NewEENode \nOutBlk=[UncTuple(3)]", color = "green", shape = "box"]; 
  1[label = "Send\nSend to: v_vmart_node0001\nNet id: 1000\nMerge\nUnc: Char(2)\nUnc: Varchar(64)\nUnc: Varchar(64)\nUnc: Varchar(64)", color = "green", shape = "box"]; 
  2[label = "Sort: (keys = A,A,N)\nUnc: Char(2)\nUnc: Varchar(64)\nUnc: Varchar(64)", color = "green", shape = "box"]; 
  3[label = "ExprEval: \n store_dimension.store_state\n store_dimension.store_city\n store_dimension.store_name\n Unc: Char(2)\nUnc: Varchar(64)\nUnc: Varchar(64)", color = "green", shape = "box"]; 
  4[label = "StorageUnionStep: store_dimension_p_b0\nUnc: Varchar(64)\nUnc: Varchar(64)\nUnc: Char(2)", color = "purple", shape = "box"]; 
  5[label = "ScanStep: store_dimension_p_b0\n store_key (not emitted)\n store_name\n store_city\n store_state\n Unc: Varchar(64)\nUnc: Varchar(64)\nUnc: Char(2)", color = "brown", shape = "box"]; 
  1->0 [label = "0",color = "blue"]; 
  2->1 [label = "0",color = "blue"]; 
  3->2 [label = "0",color = "blue"]; 
  4->3 [label = "0",color = "blue"]; 
  5->4 [label = "0",color = "blue"]; 
}
```

GraphViz renders this output as follows:
Query Plan Information and Structure

Depending on the query and database schema, EXPLAIN output includes the following information:

- Tables referenced by the statement
- Estimated costs
- Estimated row cardinality
- Path ID, an integer that links to error messages and profiling counters so you troubleshoot performance issues more easily. For more information, see Profiling Query Plans.

- Data operations such as SORT, FILTER, LIMIT, and GROUP BY
- Projections used
- Information about statistics—for example, whether they are current or out of range
- Algorithms chosen for operations into the query, such as HASH/MERGE or GROUPBY HASH/GROUPBY PIPELINED
- Data redistribution (broadcast, segmentation) across cluster nodes

Example

In the EXPLAIN output that follows, the optimizer processes a query in three steps, where each step identified by a unique path ID:

- 0: Limit
- 1: Sort
- 2: Storage access and filter
Note: A storage access operation can scan more than the columns in the SELECT list— for example, columns referenced in WHERE clause.

**Query Plan Statistics**

If you query a table whose statistics are unavailable or out-of-date, the optimizer might choose a sub-optimal query plan.

You can resolve many issues related to table statistics by calling `ANALYZE_STATISTICS`. This function let you update statistics at various scopes: one or more table columns, a single table, or all database tables.

If you update statistics and find that the query still performs sub-optimally, run your query through Database Designer and choose incremental design as the design type.

For detailed information about updating database statistics, see [Collecting Database Statistics](#).

**Statistics Hints in Query Plans**

Query plans can contain information about table statistics through two hints: `NO STATISTICS` and `STALE STATISTICS`. For example, the following query plan fragment includes `NO STATISTICS` to indicate that histograms are unavailable:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The following query plan fragment includes `STALE STATISTICS` to indicate that the predicate has fallen outside the histogram range:
Cost and Rows Path

The following EXPLAIN output shows the Cost operator:

```
| Access Path: +--SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)  
| | Output Only: 10 tuples  
| | Execute on: Query Initiator  
| | +----> SORT [Cost: 370, Rows: 544] (PATH ID: 1)  
| | | Order: customer_dimension.customer_name ASC  
| | | Output Only: 10 tuples  
| | | Execute on: Query Initiator  
| | | +----> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)  
| | | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001  
| | | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name  
| | | | Filter: (customer_dimension.customer_gender = 'Male')  
| | | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))  
| | | | Execute on: Query Initiator
```

The Row operator is the number of rows the optimizer estimates the query will return. Letters after numbers refer to the units of measure (K=thousand, M=million, B=billion, T=trillion), so the output for the following query indicates that the number of rows to return is 50 thousand.

```
=> EXPLAIN SELECT customer_gender FROM customer_dimension;
Access Path:  
+---STORAGE ACCESS for customer_dimension [Cost: 17, Rows: 50K (3 RLE)] (PATH ID: 1)  
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001  
| | Materialize: customer_dimension.customer_gender  
| | Execute on: Query Initiator
```

The reference to (3 RLE) in the STORAGE ACCESS path means that the optimizer estimates that the storage access operator returns 50K rows. Because the column is run-length encoded (RLE), the real number of RLE rows returned is only three rows:

- 1 row for female
- 1 row for male
- 1 row that represents unknown (NULL) gender

**Note:** See Query Plans for more information about how the optimizer estimates cost.
Projection Path

You can see which projections the optimizer chose for the query plan by looking at the Projection path in the textual output:

```
EXPLAIN SELECT
    customer_name,
    customer_state
FROM customer_dimension
WHERE customer_state in ('MA', 'NH')
AND customer_gender = 'Male'
ORDER BY customer_name
LIMIT 10;
Access Path:
  +-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
    |   Output Only: 10 tuples
    |   Execute on: Query Initiator
    |  >----> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
    |    |   Order: customer_dimension.customer_name ASC
    |    |   Output Only: 10 tuples
    |    |   Execute on: Query Initiator
    |  >----> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
    |    |   Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
    |    |    Materialize: customer_dimension.customer_state, customer_dimension.customer_name
    |    |    Filter: (customer_dimension.customer_gender = 'Male')
    |    |    Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
    |    |   Execute on: Query Initiator
```

The query optimizer automatically picks the best projections, but without reasonably accurate statistics, the optimizer could choose a suboptimal projection or join order for a query. For details, see Collecting Statistics.

Vertica considers which projection to choose for a plan by considering the following aspects:

- How columns are joined in the query
- How the projections are grouped or sorted
- Whether SQL analytic operations applied
- Any column information from a projection's storage on disk

As Vertica scans the possibilities for each plan, projections with the higher initial costs could end up in the final plan because they make joins cheaper. For example, a query can be answered with many possible plans, which the optimizer considers before choosing one of them. For efficiency, the optimizer uses sophisticated algorithms to prune intermediate partial plan fragments with higher cost. The optimizer knows that intermediate plan fragments might initially look bad (due to high storage access cost) but which produce excellent final plans due to other optimizations that it allows.
If your statistics are up to date but the query still performs poorly, run the query through the Database Designer. For details, see Incremental Design.

Tips

- To test different segmented projections, refer to the projection by name in the query.
- For optimal performance, write queries so the columns are sorted the same way that the projection columns are sorted.

See Also

- Reducing Query Run Time
- Creating Custom Designs
- Physical Schema

Join Path

Just like a join query, which references two or more tables, the Join step in a query plan has two input branches:

- The left input, which is the outer table of the join
- The right input, which is the inner table of the join

In the following query, the T1 table is the left input because it is on the left side of the JOIN keyword, and the T2 table is the right input, because it is on the right side of the JOIN keyword:

```sql
SELECT * FROM T1 JOIN T2 ON T1.x = T2.x;
```

Outer versus Inner Join

Query performance is better if the small table is used as the inner input to the join. The query optimizer automatically reorders the inputs to joins to ensure that this is the case unless the join in question is an outer join.

Note: If the configuration parameter EnableForceOuter is set to 1, you can control join inputs for specific tables through ALTER TABLE .. FORCE OUTER. For details, see Controlling Join Inputs in Analyzing Data.
The following example shows a query and its plan for a left outer join:

```sql
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
  -> FROM online_sales.online_sales_fact OSI
  -> LEFT OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
  +--JOIN HASH [LeftOuter] [Cost: 4K, Rows: 5M] (PATH ID: 1)
    |  Join Cond: (CD.customer_key = OSI.customer_key)
    |  Materialize at Output: OSI.sale_date_key
    |  Execute on: All Nodes
    |  +-- Outer -- STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
    |  |  Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
    |  |  Materialize: OSI.customer_key
    |  |  Execute on: All Nodes
    |  +-- Inner --> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
    |  |  Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
    |  |  Materialize: CD.annual_income, CD.customer_key
    |  |  Execute on: All Nodes
```

The following example shows a query and its plan for a full outer join:

```sql
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
  -> FROM online_sales.online_sales_fact OSI
  -> FULL OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
  +--JOIN HASH [FullOuter] [Cost: 18K, Rows: 5M] (PATH ID: 1) Outer (RESEGMENT) Inner (FILTER)
    |  Join Cond: (CD.customer_key = OSI.customer_key)
    |  Execute on: All Nodes
    |  +-- Outer -- STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
    |  |  Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
    |  |  Materialize: OSI.sale_date_key, OSI.customer_key
    |  |  Execute on: All Nodes
    |  +-- Inner --> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
    |  |  Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
    |  |  Materialize: CD.annual_income, CD.customer_key
    |  |  Execute on: All Nodes
```

Hash and Merge Joins

Vertica has two join algorithms to choose from: merge join and hash join. The optimizer automatically chooses the most appropriate algorithm, given the query and projections in a system.

For the following query, the optimizer chooses a hash join.

```sql
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
  -> FROM online_sales.online_sales_fact OSI
  -> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
  +--JOIN HASH [Cost: 4K, Rows: 5M] (PATH ID: 1)
    |  Join Cond: (CD.customer_key = OSI.customer_key)
    |  Materialize at Output: OSI.sale_date_key
    |  Execute on: All Nodes
    |  +-- Outer --> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
    |  |  Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_design
```
Tip: If you get a hash join when you are expecting a merge join, it means that at least one of the projections is not sorted on the join column (for example, `customer_key` in the preceding query). To facilitate a merge join, you might need to create different projections that are sorted on the join columns.

In the next example, the optimizer chooses a merge join. The optimizer's first pass performs a merge join because the inputs are presorted, and then it performs a hash join.

```sql
=> EXPLAIN SELECT count(*) FROM online_sales.online_sales_fact OSI
  -> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key
  -> INNER JOIN product_dimension PD ON PD.product_key = OSI.product_key;
Access Path:
  +GROUPBY NOTHING [Cost: 8K, Rows: 1] (PATH ID: 1)
  | Aggregates: count(*)
  | Execute on: All Nodes
  |----- JOIN HASH [Cost: 7K, Rows: 5M] (PATH ID: 2)
  |  Join Cond: (PD.product_key = OSI.product_key)
  |  Materialize at Input: OSI.product_key
  |  Execute on: All Nodes
  |----- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 4K, Rows: 5M] (PATH ID: 3)
  |  Join Cond: (CD.customer_key = OSI.customer_key)
  |  Execute on: All Nodes
  |----- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 4)
  |  Projection: online_sales.online_sales_fact_DBD_12_rep_vmartdb_design_vmartdb_design
  |  Materialize: OSI.customer_key
  |  Execute on: All Nodes
  |----- Inner -> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 5)
  |  Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
  |  Materialize: CD.customer_key
  |  Execute on: All Nodes
  |----- Inner -> STORAGE ACCESS for PD [Cost: 152, Rows: 60K] (PATH ID: 6)
  |  Projection: public.product_dimension_DBD_2_rep_vmartdb_design_vmartdb_design_node0001
  |  Materialize: PD.product_key
  |  Execute on: All Nodes
```

**Inequality Joins**

Vertica joins with equality predicates very efficiently. The query plan shows equality join predicates as join condition (Join Cond).

```sql
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
  -> FROM online_sales.online_sales_fact OSI
  -> INNER JOIN customer_dimension CD
  -> ON CD.customer_key = OSI.customer_key;
Access Path:
```

---

**Vertica Analytic Database (9.0.x)**

Page 825 of 6180
However, inequality joins are treated like cross joins and can run less efficiently, which you can see by the change in cost between the two queries:

```
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
  -> FROM online_sales.online_sales_fact OSI
  -> INNER JOIN customer_dimension CD
  -> ON CD.customer_key < OSI.customer_key;
Access Path:
  +JOIN HASH [Cost: 98M, Rows: 5M] (PATH ID: 1)
  | Join Filter: (CD.customer_key < OSI.customer_key)
  | Materialize at Output: CD.annual_income
  | Execute on: All Nodes
  | ++ Outer --> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 2)
  |  | Projection: public.customer_dimension_DB0_1_rep_vmartdb_design_vmartdb_design_node0001
  |  | Materialize: CD.customer_key
  |  | Execute on: All Nodes
  | ++ Inner --> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 3)
  |  | Projection: online_sales.online_sales_fact_DB0_12_seg_vmartdb_design_vmartdb_design
  |  | Materialize: OSI.sale_date_key, OSI.customer_key
  |  | Execute on: All Nodes
```

Event Series Joins

Event series joins are denoted by the INTERPOLATED path.

```
=> EXPLAIN SELECT * FROM hTicks h FULL OUTER JOIN aTicks a -> ON (h.time INTERPOLATE PREVIOUS
Access Path:
  +JOIN (INTERPOLATED) [FullOuter] [Cost: 31, Rows: 4 (NO STATISTICS)] (PATH ID: 1)
  | Outer (SORT ON JOIN KEY) Inner (SORT ON JOIN KEY)
  | Join Cond: (h."time" = a."time")
  | Execute on: Query Initiator
  | ++ Outer --> STORAGE ACCESS for h [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 2)
  |  | Projection: public.hTicks_node0004
  |  | Materialize: h.stock, h."time", h.price
  |  | Execute on: Query Initiator
  | ++ Inner --> STORAGE ACCESS for a [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 3)
  |  | Projection: public.aTicks_node0004
  |  | Materialize: a.stock, a."time", a.price
  |  | Execute on: Query Initiator
```
Path ID

The PATH ID is a unique identifier that Vertica assigns to each operation (path) within a query plan. The same identifier is shared by:

- Query plans
- Join error messages
- System tables EXECUTION_ENGINE_PROFILES and QUERY_PLAN_PROFILES

Path IDs can help you trace issues to their root cause. For example, if a query returns a join error, preface the query with EXPLAIN and look for PATH ID n in the query plan to see which join in the query had the problem.

For example, the following EXPLAIN output shows the path ID for each path in the optimizer's query plan:

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN on y=z;
Access Path:
  +--JOIN MERGEJOIN(inputs presorted) [Cost: 815, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
  |  Join Cond: (dim.y = ext.z)
  |  Materialize at Output: fact.x
  |  Execute on: All Nodes
  |  ++-- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 408, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
  |  |  Join Cond: (fact.x = dim.y)
  |  |  Execute on: All Nodes
  |  |  ++-- Outer -> STORAGE ACCESS for fact [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
  |  |  |  Projection: public.fact_super
  |  |  |  |  Materialize: fact.x
  |  |  |  |  |  Execute on: All Nodes
  |  |  |  |  ++-- Inner -> STORAGE ACCESS for dim [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 4)
  |  |  |  |  |  Projection: public.dim_super
  |  |  |  |  |  Materialize: dim.y
  |  |  |  |  |  |  Execute on: All Nodes
  |  |  |  |  ++-- Inner -> STORAGE ACCESS for ext [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 5)
  |  |  |  |  |  |  Projection: public.ext_super
  |  |  |  |  |  |  Materialize: ext.z
  |  |  |  |  |  |  Execute on: All Nodes
```

Filter Path

The Filter step evaluates predicates on a single table. It accepts a set of rows, eliminates some of them (based on the criteria you provide in your query), and returns the rest. For example, the optimizer can filter local data of a join input that will be joined with another re-segmented join input.
The following statement queries the customer_dimension table and uses the WHERE clause to filter the results only for male customers in Massachusetts and New Hampshire.

```
EXPLAIN SELECT
   CD.customer_name,
   CD.customer_state,
   AVG(CD.customer_age) AS avg_age,
   COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH') AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name;
```

The query plan output is as follows:

```
Access Path:
  +--GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 1)
     | Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
     | Group By: CD.customer_state, CD.customer_name
     | Execute on: Query Initiator
     | +---- STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 2)
        | | Projection: public.customer_dimension_DB0_1_rep_vmartdb_design_vmartdb_design_node0001
        | | Materialize: CD.customer_state, CD.customer_name, CD.customer_age
        | | Filter: (CD.customer_gender = 'Male')
        | | Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
        | | Execute on: Query Initiator
```

GROUP BY Paths

A GROUP BY operation has two algorithms:

- **GROUPBY HASH** input is not sorted by the group columns, so Vertica builds a hash table on those group columns in order to process the aggregates and group by expressions.

- **GROUPBY PIPELINED** requires that inputs be presorted on the columns specified in the group, which means that Vertica need only retain data in the current group in memory. GROUPBY PIPELINED operations are preferred because they are generally faster and require less memory than GROUPBY HASH. GROUPBY PIPELINED is especially useful for queries that process large numbers of high-cardinality group by columns or DISTINCT aggregates.

If possible, the query optimizer chooses the faster algorithm GROUPBY PIPELINED over GROUPBY HASH.

**Note:** For details, see GROUP BY Implementation Options in Analyzing Data.

GROUPBY HASH Query Plan

Here's an example of how GROUPBY HASH operations look in EXPLAIN output.
=> EXPLAIN SELECT COUNT(DISTINCT annual_income) 
    FROM customer_dimension 
    WHERE customer_region='NorthWest';

The output shows that the optimizer chose the less efficient GROUPBY HASH path, which means the projection was not presorted on the annual_income column. If such a projection is available, the optimizer would choose the GROUPBY PIPELINED algorithm.

GROUPBY PIPELINED Query Plan

If you have a projection that is already sorted on the customer_gender column, the optimizer chooses the faster GROUPBY PIPELINED operation:

=> EXPLAIN SELECT count(distinct customer_gender) from customer_dimension;

Access Path: 
+--GROUPBY NOTHING [Cost: 22, Rows: 1] (PATH ID: 1) 
 | | Aggregates: count(DISTINCT customer_dimension.customer_gender) 
 | | Execute on: Query Initiator 
 | | +-- GROUPBY PIPELINED [Cost: 20, Rows: 10K] (PATH ID: 2) 
 | | | | Group By: customer_dimension.customer_gender 
 | | | | Execute on: Query Initiator 
 | | | | +-- STORAGE ACCESS for customer_dimension [Cost: 17, Rows: 50K (3 RLE)] (PATH ID: 3) 
 | | | | | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001 
 | | | | | | Materialize: customer_dimension.customer_gender 
 | | | | | | Execute on: Query Initiator

Similarly, the use of an equality predicate, such as in the following query, preserves GROUPBY PIPELINED:

=> EXPLAIN SELECT COUNT(DISTINCT annual_income) FROM customer_dimension 
    WHERE customer_gender = 'Female';

Access Path: +--GROUPBY NOTHING [Cost: 161, Rows: 1] (PATH ID: 1) 
 | | Aggregates: count(DISTINCT customer_dimension.annual_income) 
 | | +-- GROUPBY PIPELINED [Cost: 158, Rows: 10K] (PATH ID: 2) 
 | | | | Group By: customer_dimension.annual_income 
 | | | | +-- STORAGE ACCESS for customer_dimension [Cost: 144, Rows: 47K] (PATH ID: 3) 
 | | | | | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001 
 | | | | | | Materialize: customer_dimension.annual_income 
 | | | | | | Filter: (customer_dimension.customer_gender = 'Female')
Tip: If EXPLAIN reports GROUPBY HASH, modify the projection design to force it to use GROUPBY PIPELINED.

Sort Path

The SORT operator sorts the data according to a specified list of columns. The EXPLAIN output indicates the sort expressions and if the sort order is ascending (ASC) or descending (DESC).

For example, the following query plan shows the column list nature of the SORT operator:

```sql
EXPLAIN SELECT
    CD.customer_name,
    CD.customer_state,
    AVG(CD.customer_age) AS avg_age,
    COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH')
    AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name
ORDER BY avg_age, customer_name;
Access Path:
  +---- SORT [Cost: 422, Rows: 544] (PATH ID: 1)
  |     Order: (<SVAR> / float8(<SVAR>)) ASC, CD.customer_name ASC
  |     Execute on: Query Initiator
  |     ----> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)
  |     |     Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
  |     |     Group By: CD.customer_state, CD.customer_name
  |     |     Execute on: Query Initiator
  |     |     ----> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 3)
  |     |     |     Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
  |     |     |     Materialize: CD.customer_state, CD.customer_name, CD.customer_age
  |     |     |     |     Filter: (CD.customer_gender = 'Male')
  |     |     |     |     Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
  |     |     |     |     Execute on: Query Initiator
```

If you change the sort order to descending, the change appears in the query plan:

```sql
EXPLAIN SELECT
    CD.customer_name,
    CD.customer_state,
    AVG(CD.customer_age) AS avg_age,
    COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH')
    AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name
ORDER BY avg_age DESC, customer_name;
Access Path:
  +---- SORT [Cost: 422, Rows: 544] (PATH ID: 1)
  |     Order: (<SVAR> / float8(<SVAR>)) DESC, CD.customer_name ASC
  |     Execute on: Query Initiator
  |     ----> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)
  |     |     Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
  |     |     Group By: CD.customer_state, CD.customer_name
```

Vertica Analytic Database (9.0.x)
Limit Path

The LIMIT path restricts the number of result rows based on the LIMIT clause in the query. Using the LIMIT clause in queries with thousands of rows might increase query performance.

The optimizer pushes the LIMIT operation as far down as possible in queries. A single LIMIT clause in the query can generate multiple Output Only plan annotations.

Data Redistribution Path

The optimizer can redistribute join data in two ways:

- Broadcasting
- Resegmentation

Broadcasting

Broadcasting sends a complete copy of an intermediate result to all nodes in the cluster. Broadcast is used for joins in the following cases:

- One table is very small (usually the inner table) compared to the other.
- Vertica can avoid other large upstream resegmentation operations.
Outer join or subquery semantics require one side of the join to be replicated.

For example:

```sql
=> EXPLAIN SELECT * FROM T1 LEFT JOIN T2 ON T1.a > T2.y;
Access Path:
+-JOIN HASH [LeftOuter] [Cost: 40K, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (BROADCAST)
  | Join Filter: (T1.a > T2.y)
  | Materialize at Output: T1.b
  | Execute on: All Nodes
  | +- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
  |   | Projection: public.T1_b0
  |   | Materialize: T1.a
  |   | Execute on: All Nodes
  | +- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
  |   | Projection: public.T2_b0
  |   | Materialize: T2.x, T2.y
  |   | Execute on: All Nodes
```

Resegmentation

Resegmentation takes an existing projection or intermediate relation and resegments the data evenly across all cluster nodes. At the end of the resegmentation operation, every row from the input relation is on exactly one node. Resegmentation is the operation used most often for distributed joins in Vertica if the data is not already segmented for local joins. For more detail, see Identical Segmentation in Analyzing Data.

For example:

```sql
=> CREATE TABLE T1 (a INT, b INT) SEGMENTED BY HASH(a) ALL NODES;
=> CREATE TABLE T2 (x INT, y INT) SEGMENTED BY HASH(x) ALL NODES;
=> EXPLAIN SELECT * FROM T1 JOIN T2 ON T1.a = T2.y;

------------------------------------------- QUERY PLAN DESCRIPTION: -------------------------------------------
Access Path:
+-JOIN HASH [Cost: 639, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (RESEGMENT)
  | Join Cond: (T1.a = T2.y)
  | Materialize at Output: T1.b
  | Execute on: All Nodes
  | +- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
  |   | Projection: public.T1_b0
  |   | Materialize: T1.a
  |   | Execute on: All Nodes
  | +- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
  |   | Projection: public.T2_b0
  |   | Materialize: T2.x, T2.y
  |   | Execute on: All Nodes
```
Analytic Function Path

Vertica attempts to optimize multiple SQL-99 Analytic Functions from the same query by grouping them together in Analytic Group areas.

For each analytical group, Vertica performs a distributed sort and resegment of the data, if necessary.

You can tell how many sorts and resegments are required based on the query plan.

For example, the following query plan shows that the FIRST_VALUE and LAST_VALUE functions are in the same analytic group because their OVER clause is the same. In contrast, ROW_NUMBER() has a different ORDER BY clause, so it is in a different analytic group. Because both groups share the same PARTITION BY deal_stage clause, the data does not need to be resegmented between groups:

```
EXPLAIN SELECT
  first_value(deal_size) OVER (PARTITION BY deal_stage
  ORDER BY deal_size),
  last_value(deal_size) OVER (PARTITION BY deal_stage
  ORDER BY deal_size),
  row_number() OVER (PARTITION BY deal_stage
  ORDER BY largest_bill_amount)
FROM customer_dimension;
```

Access Path:
+-ANALYTICAL [Cost: 1K, Rows: 50K] (PATH ID: 1)
 | Analytic Group
 | Group Sort: customer_dimension.deal_stage ASC, customer_dimension.largest_bill_amount ASC NULLS
LAST
 | Analytic Group
 | Functions: first_value(), last_value()
 | Group Filter: customer_dimension.deal_stage
 | Group Sort: customer_dimension.deal_stage ASC, customer_dimension.deal_size ASC NULL LAST
 | Execute on: All Nodes
| +----- STORAGE ACCESS for customer_dimension [Cost: 263, Rows: 50K] (PATH ID: 2)
 | | Projection: public.customer_dimension_DB0_1_rep_vmart_vmart_node0001
 | | Materialize: customer_dimension.largest_bill_amount,
 | | customer_dimension.deal_stage, customer_dimension.deal_size
 | | Execute on: All Nodes

See Also

Invoking Analytic Functions
Node Down Information

Vertica provides performance optimization when cluster nodes fail by distributing the work of the down nodes uniformly among available nodes throughout the cluster.

When a node in your cluster is down, the query plan identifies which node the query will execute on. To help you quickly identify down nodes on large clusters, EXPLAIN output lists up to six nodes, if the number of running nodes is less than or equal to six, and lists only down nodes if the number of running nodes is more than six.

Note: The node that executes down node queries is not always the same one.

The following table provides more detail:

<table>
<thead>
<tr>
<th>Node state</th>
<th>EXPLAIN output</th>
</tr>
</thead>
<tbody>
<tr>
<td>If all nodes are up, EXPLAIN output indicates All Nodes.</td>
<td>Execute on: All Nodes</td>
</tr>
<tr>
<td>If fewer than 6 nodes are up, EXPLAIN lists up to six running nodes.</td>
<td>Execute on: [node_list].</td>
</tr>
<tr>
<td>If more than 6 nodes are up, EXPLAIN lists only non-running nodes.</td>
<td>Execute on: All Nodes Except [node_list]</td>
</tr>
<tr>
<td>If the node list contains non-ephemeral nodes, the EXPLAIN output indicates All Permanent Nodes.</td>
<td>Execute on: All Permanent Nodes</td>
</tr>
<tr>
<td>If the path is being run on the query initiator, the EXPLAIN output indicates Query Initiator.</td>
<td>Execute on: Query Initiator</td>
</tr>
</tbody>
</table>

Examples

In the following example, the down node is v_vmart_node0005, and the node v_vmart_node0006 will execute this run of the query.

```sql
=> EXPLAIN SELECT * FROM test;
QUERY PLAN

-------------------------------
QUERY PLAN DESCRIPTION:
-------------------------------
EXPLAIN SELECT * FROM mytable;
Access Path:
   +--STORAGE ACCESS for mytable [Cost: 10, Rows: 2] (PATH ID: 1)
      | Projection: public.mytable_b0
```
The All Permanent Nodes output in the following example fragment denotes that the node list is for permanent (non-ephemeral) nodes only:

```sql
=> EXPLAIN SELECT * FROM my2table;
Access Path:
  +--STORAGE ACCESS for my2table [Cost: 18, Rows: 6 (NO STATISTICS)] (PATH ID: 1)
      | Projection: public.my2table_b0
      | Materialize: my2table.x, my2table.y, my2table.z
      | Execute on: All Permanent Nodes
```

**MERGE Path**

Vertica prepares an optimized query plan for a `MERGE` statement if the statement and its tables meet the criteria described in MERGE Optimization.

Use the `EXPLAIN` keyword to determine whether Vertica can produce an optimized query plan for a given MERGE statement. If optimization is possible, the EXPLAIN-generated output contains a `[Semi]` path, as shown in the following sample fragment:

```sql
...
Access Path:
  +--DML DELETE [Cost: 0, Rows: 0]
      | Target Projection: public.A_b1 (DELETE ON CONTAINER)
      | Target Prep:
      | Execute on: All Nodes
      | +---> JOIN MERGEJOIN(inputs presorted) [Semi] [Cost: 6, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
          | Inner (RESEGMENT)
          |      | Join Cond: (A.a1 = VAL(2))
          |      | Execute on: All Nodes
          |      | +-- Outer -> STORAGE ACCESS for A [Cost: 2, Rows: 2 (NO STATISTICS)] (PATH ID: 2)
          |      | ...
```

Conversely, if Vertica cannot create an optimized plan, EXPLAIN-generated output contains RightOuter path:

```sql
...
Access Path: +--DML MERGE
      | Target Projection: public.locations_b1
      | Target Projection: public.locations_b0
      | Target Prep:
      | Execute on: All Nodes
      | +---> JOIN MERGEJOIN(inputs presorted) [RightOuter] [Cost: 28, Rows: 3 (NO STATISTICS)] (PATH ID: 1)
          | Outer (RESEGMENT) Inner (RESEGMENT)
          |      | Join Cond: (locations.user_id = VAL(2)) AND (locations.location_x = VAL(2)) AND
```

Vertica Analytic Database (9.0.x)  Page 835 of 6180
Directed Queries

Directed queries encapsulate information that the optimizer can use to create a query plan. Directed queries can serve the following goals:

- Preserve current query plans before a scheduled upgrade. In most instances, queries perform more efficiently after a Vertica upgrade. In the few cases where this is not so, you can use directed queries that you created before upgrading, to recreate query plans from the earlier version.

- Enable you to create query plans that improve optimizer performance. Occasionally, you might want to influence the optimizer to make better choices in executing a given query. For example, you can choose a different projection, or force a different join order. In this case, you can use a directed query to create a query plan that preempts any plan that the optimizer might otherwise create.

- Redirect an input query to a query that uses different semantics—for example, map a join query to a SELECT statement that queries a flattened table.

Directed Query Components

A directed query pairs two components:

- Input query: A query that triggers use of this directed query when it is active.

- Annotated query: A SQL statement with embedded optimizer hints, which instruct the optimizer how to create a query plan for the specified input query. These hints specify important query plan elements, such as join order and projection choices.

Tip: You can also use most optimizer hints directly in vsql. For information about these and other hints, see Hints in the SQL Reference Manual.

Vertica provides two methods for creating directed queries:
The optimizer can generate an annotated query from a given input query and pair the two as a directed query.

You can write your own annotated query and pair it with an input query.

For a description of both methods, see Creating Directed Queries.

Creating Directed Queries

CREATE DIRECTED QUERY associates an input query with a query annotated with optimizer hints. It stores the association under a unique identifier. CREATE DIRECTED QUERY has two variants:

- CREATE DIRECTED QUERY OPTIMIZER directs the query optimizer to generate annotated SQL from the specified input query. The annotated query contains hints that the optimizer can use to recreate its current query plan for that input query.

- CREATE DIRECTED QUERY CUSTOM specifies an annotated query supplied by the user. Vertica associates the annotated query with the input query specified by the last SAVE QUERY statement.

In both cases, Vertica associates the annotated query and input query, and registers their association in the system table V_CATALOG.DIRECTED_ QUERIES under query_name. The two approaches can be used together: you can use the annotated SQL that the optimizer creates as the basis for creating your own (custom) directed queries.

Optimizer-Generated Directed Queries

CREATE DIRECTED QUERY OPTIMIZER passes an input query to the optimizer, which generates an annotated query from its own query plan. It then pairs the input and annotated queries and saves them as a directed query.

Note: The input query that you supply for optimizer-generated directed queries supports only one optimizer hint, IGNORECONST.

You can use optimized-generated directed queries to capture query plans before you upgrade. Doing so can be especially useful if you detect diminished performance of a given query after the upgrade. In this case, you can use the corresponding directed query to recreate an earlier query plan, and compare its performance to the plan generated by the current optimizer.
For example, the following SQL statements create and activate the directed query `findBostonCashiers_OPT`:

```sql
=> CREATE DIRECTED QUERY OPTIMIZER 'findBostonCashiers_OPT'
    SELECT employee_first_name, employee_last_name FROM public.employee_dimension
    WHERE employee_city='Boston' and job_title='Cashier';
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
ACTIVATE DIRECTED QUERY
```

After this directed query plan is activated, the optimizer uses it to generate a query plan for all subsequent invocations of its input query. You can view the optimizer-generated annotated query by either calling `GET DIRECTED QUERY` or querying the system table `V_CATALOG.DIRECTED_QUERIES`:

```sql
=> SELECT query_name, annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name = 'findBostonCashiers_OPT';
-| query_name         | annotated_query                                                                 |
-|-------------------|---------------------------------------------------------------------------------|
-| findBostonCashiers_OPT | SELECT /*+verbatim*/ employee_dimension.employee_first_name AS employee_first_name, employee_last_name AS employee_last_name FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/ WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7)) |
```

The annotated SQL includes two hints:

- `/*+verbatim*/` specifies to execute the annotated query exactly as written and produce a query plan accordingly.
- `/*+projs('public.Emp_Dimension')*/` directs the optimizer to create a query plan that uses the projection `public.Emp_Dimension`.

The following `EXPLAIN` statement verifies the optimizer's use of this directed query and the specified projection:

```sql
<table>
<thead>
<tr>
<th>QUERY PLAN DESCRIPTION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension WHERE employee_city='Boston' AND job_title='Cashier';</td>
</tr>
</tbody>
</table>

The following active directed query(query name: findBostonCashiers_OPT) is being executed:

SELECT /*+verbatim*/ employee_dimension.employee_first_name, employee_dimension.employee_last_name FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/ WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7))

Access Path:
+STORAGE ACCESS for employee_dimension [Cost: 60, Rows: 10K (NO STATISTICS)] (PATH ID: 1) |
  | Projection: public.employee_dimension_bB |
  | Materialize: employee_dimension.employee_first_name, employee_dimension.employee_last_name |
  | Filter: (employee_dimension.employee_city = 'Boston') |
```

"For example, the following SQL statements create and activate the directed query `findBostonCashiers_OPT`:

```sql
=> CREATE DIRECTED QUERY OPTIMIZER 'findBostonCashiers_OPT'
    SELECT employee_first_name, employee_last_name FROM public.employee_dimension
    WHERE employee_city='Boston' and job_title='Cashier';
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
ACTIVATE DIRECTED QUERY
```

After this directed query plan is activated, the optimizer uses it to generate a query plan for all subsequent invocations of its input query. You can view the optimizer-generated annotated query by either calling `GET DIRECTED QUERY` or querying the system table `V_CATALOG.DIRECTED_QUERIES`:

```sql
=> SELECT query_name, annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name = 'findBostonCashiers_OPT';
-| query_name         | annotated_query                                                                 |
-|-------------------|---------------------------------------------------------------------------------|
-| findBostonCashiers_OPT | SELECT /*+verbatim*/ employee_dimension.employee_first_name AS employee_first_name, employee_last_name AS employee_last_name FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/ WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7)) |
```

The annotated SQL includes two hints:

- `/*+verbatim*/` specifies to execute the annotated query exactly as written and produce a query plan accordingly.
- `/*+projs('public.Emp_Dimension')*/` directs the optimizer to create a query plan that uses the projection `public.Emp_Dimension`.

The following `EXPLAIN` statement verifies the optimizer's use of this directed query and the specified projection:

```sql
<table>
<thead>
<tr>
<th>QUERY PLAN DESCRIPTION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension WHERE employee_city='Boston' AND job_title='Cashier';</td>
</tr>
</tbody>
</table>

The following active directed query(query name: findBostonCashiers_OPT) is being executed:

SELECT /*+verbatim*/ employee_dimension.employee_first_name, employee_dimension.employee_last_name FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/ WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7))

Access Path:
+STORAGE ACCESS for employee_dimension [Cost: 60, Rows: 10K (NO STATISTICS)] (PATH ID: 1) |
  | Projection: public.employee_dimension_bB |
  | Materialize: employee_dimension.employee_first_name, employee_dimension.employee_last_name |
  | Filter: (employee_dimension.employee_city = 'Boston') |
```
Custom Directed Queries

CREATE DIRECTED QUERY CUSTOM specifies an annotated query and pairs it to an input query previously saved by SAVE QUERY. The SAVE QUERY statement must precede CREATE DIRECTED QUERY CUSTOM. SAVE QUERY temporarily saves an input query for use in creating a directed query. You must issue both statements in the same user session.

Note: The input query that you supply to SAVE QUERY supports only one optimizer hint, IGNORECONST.

In the following example, SAVE QUERY saves a query. The CREATE DIRECTED QUERY CUSTOM statement that follows it provides an annotated query that includes a /*+projs*/ hint. This hint instructs the optimizer to use the projection public.Emp_Dimension_Unseg:

```sql
=> SAVE QUERY SELECT employee_first_name, employee_last_name FROM employee_dimension
    WHERE employee_city='Boston' AND job_title='Cashier';
SAVE QUERY

=> CREATE DIRECTED QUERY CUSTOM 'findBostonCashiers_CUSTOM'
    SELECT employee_first_name, employee_last_name
    FROM employee_dimension /*+Projs('public.emp_dimension_unseg')*/
    WHERE employee_city='Boston' AND job_title='Cashier';
CREATE DIRECTED QUERY
```

Caution: Vertica associates a saved query and annotated query without checking whether the input and annotated queries are compatible. Be careful to sequence SAVE QUERY and CREATE DIRECTED QUERY CUSTOM so the saved and directed queries are correctly matched.

After this directed query plan is activated, the optimizer uses it to generate a query plan for all subsequent invocations of its input query. The following EXPLAIN output verifies the optimizer’s use of this directed query and the projection it specifies:

```sql
=> DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
DEACTIVATE DIRECTED QUERY
=> ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
ACTIVATE DIRECTED QUERY

=> EXPLAIN SELECT employee_first_name, employee_last_name FROM employee_dimension
    WHERE employee_city='Boston' AND job_title='Cashier';
QUERY PLAN
```
Using Optimizer-Generated and Custom Directed Queries Together

You can use the annotated SQL that the optimizer creates as the basis for creating your own (custom) directed queries. This approach can be especially useful in evaluating the plan that the optimizer creates to handle a given query, and testing plan modifications.

For example, you might want to modify how the optimizer implements the following query:

```sql
SELECT COUNT(customer_name) Total, customer_region Region
FROM (store_sales s JOIN customer_dimension c ON c.customer_key = s.customer_key)
JOIN product_dimension p ON s.product_key = p.product_key
WHERE p.category_description ilike '%Medical%'
  AND p.product_description ilike '%antibiotics%'
  AND c.customer_age <= 30 AND YEAR(s.sales_date)>2017
GROUP BY customer_region;
```

When you run EXPLAIN on this query, you discover that the optimizer uses projection customers_proj_age for the customer_dimension table. This projection is sorted on column customer_age. Consequently, the optimizer hash-joins the tables store_sales and customer_dimension on customer_key.

After analyzing customer_dimension table data, you observe that most customers are under 30, so it makes more sense to use projection customer_proj_id for the customer_dimension table, which is sorted on customer_key:

You can create a directed query that encapsulates this change as follows:
1. Obtain optimizer-generated annotations on the query with `EXPLAIN ANNOTATED`:

```
=> \o annotatedQuery
=> EXPLAIN ANNOTATED SELECT COUNT(customer_name) Total, customer_region Region
    FROM (store_sales s JOIN customer_dimension c ON c.customer_key = s.customer_key)
    JOIN product_dimension p ON s.product_key = p.product_key
    WHERE p.category_description ilike 'Medical%'
        AND p.product_description ilike 'antibiotics'
        AND c.customer_age <= 30 AND YEAR(s.sales_date)=2017
    GROUP BY customer_region;
=> \o
=> \! cat annotatedQuery
...
SELECT /*+syntactic_join,verbatim*/ count(c.customer_name) AS Total, c.customer_region AS Region
    FROM ((public.store_sales AS s/*+projs('public.store_sales_super')*/
        JOIN /*+Distrib(L,B),JType(H)*/ public.customer_dimension AS c/*+projs('public.customers_proj_id')*/
            ON (c.customer_key = s.customer_key))
        JOIN /*+Distrib(L,B),JType(H)*/ public.product_dimension AS p/*+projs('public.product_dimension')*/
            ON (s.product_key = p.product_key))
    WHERE ((date_part('year':varchar(4), (s.sales_date)::timestamp(0))::int = 2017)
        AND (c.customer_age <= 30)
        AND ((p.category_description)::varchar(32) ~~* 'Medical%':varchar(9))
        AND (p.product_description ~~* 'antibiotics%':varchar(13))
GROUP BY /*+GByType(Hash)*/ 2
(4 rows)
```

2. Modify the annotated query:

```
SELECT /*+syntactic_join,verbatim*/ count(c.customer_name) AS Total, c.customer_region AS Region
    FROM ((public.store_sales AS s/*+projs('public.store_sales_super')*/
        JOIN /*+Distrib(L,B),JType(H)*/ public.customer_dimension AS c/*+projs('public.customer_dimension')*/
            ON (c.customer_key = s.customer_key))
        JOIN /*+Distrib(L,B),JType(H)*/ public.product_dimension AS p/*+projs('public.product_dimension')*/
            ON (s.product_key = p.product_key))
    WHERE ((date_part('year':varchar(4), (s.sales_date)::timestamp(0))::int = 2017)
        AND (c.customer_age <= 30)
        AND ((p.category_description)::varchar(32) ~~* 'Medical%':varchar(9))
        AND (p.product_description ~~* 'antibiotics%':varchar(13))
GROUP BY /*+GByType(Hash)*/ 2
```

3. Use the modified annotated query to create the desired directed query:

   - Save the desired input query with `SAVE QUERY`:

```
=> SAVE QUERY SELECT COUNT(customer_name) Total, customer_region Region
    FROM (store_sales s JOIN customer_dimension c ON c.customer_key = s.customer_key)
    JOIN product_dimension p ON s.product_key = p.product_key
    WHERE p.category_description ilike 'Medical'
        AND p.product_description ilike 'antibiotics'
```
AND c.customer_age <= 30 AND YEAR(s.sales_date)=2017
GROUP BY customer_region;

- Create a custom directed query that associates the saved input query with the modified annotated query:

=> CREATE DIRECTED QUERY CUSTOM 'getCustomersUnder31'
   SELECT /*+syntactic_join,verbatim*/ count(c.customer_name) AS Total, c.customer_region AS Region
   FROM ((public.store_sales AS s/*+projs('public.store_sales_super')*/
     JOIN /*+Distrib(L,B),JType(H)*/
       public.customer_dimension AS c/*+projs('public.customer_dimension')*/
     ON (c.customer_key = s.customer_key))
   JOIN /*+Distrib(L,B),JType(H)*/
       public.product_dimension AS p/*+projs('public.product_dimension')*/
   ON (s.product_key = p.product_key)
WHERE ((date_part('year'::varchar(4), (s.sales_date)::timestamp(0)))::int = 2017)
   AND (c.customer_age <= 30)
   AND ((p.category_description)::varchar(32) ~~* '%Medical%'::varchar(9))
   AND (p.product_description ~~* '%antibiotics%'::varchar(13))
GROUP BY /*+GByType(Hash)*/ 2;
CREATE DIRECTED QUERY

4. Activate this directed query:

ACTIVATE DIRECTED QUERY getCustomersUnder31;
ACTIVATE DIRECTED QUERY

When the optimizer processes a query that matches this directed query's input query, it uses the directed query's annotated query to generate a query plan:

EXPLAIN SELECT COUNT(customer_name) Total, customer_region Region
FROM (store_sales s JOIN customer_dimension c ON c.customer_key = s.customer_key)
JOIN product_dimension p ON s.product_key = p.product_key
WHERE p.category_description ilike '%Medical%'
   AND p.product_description ilike '%antibiotics%'
   AND c.customer_age <= 30 AND YEAR(s.sales_date)=2017
GROUP BY customer_region;

The following active directed query(query name: getCustomersUnder31) is being executed: ...

Setting Hints in Annotated Queries

The hints in a directed query's annotated query provide the Vertica optimizer instructions how to execute an input query. Annotated queries support the following hints:
Join hints specify join order, join type, and join data distribution: `SYNTACTIC_JOIN`, `DISTRIB`, `JTYPE`, `UTYPE`.

Table hints specify which projections to include and exclude in the query plan: `PROJS`, `SKIP_PROJS`.

`IGNORECONST` is supported for a directed query's input and annotated queries. For more information, see Ignoring Constants in Directed Queries.

`VERBATIM` enforces execution of an annotated query exactly as written.

Other hints in annotated queries such as `DIRECT` or `LABEL` are ignored.

You can use hints in a `vsql` query the same as in an annotated query, with two exceptions: `IGNORECONST` and `VERBATIM` are invalid for use in `vsql`. For general information about using hints, see Hints in the SQL Reference Manual.

## Ignoring Constants in Directed Queries

The `IGNORECONST` hint lets you create directed queries that support input queries with various conditions. For example, you might want to use the same directed query for the following input queries:

```sql
=> SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE Employee_city='Boston' and Employee_position='Cashier';
```

```sql
=> SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE Employee_city='Chicago' and Employee_position='Greeter';
```

In this case, you can create a directed query where input and annotated queries qualify the settings for `Employee_city` and `Employee_position` with `IGNORECONST` hints:

```sql
=> SAVE QUERY SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension
    WHERE Employee_city='somewhere'/*+IGNORECONST(1)*/
    AND Employee_position='somejob' /*+IGNORECONST(2)*;
SAVE QUERY
```

```sql
=> CREATE DIRECTED QUERY CUSTOM 'findEmployees'
    SELECT Employee_first_name, Employee_last_name
    FROM EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/
    WHERE Employee_city='somewhere'/*+IGNORECONST(1)*/
    AND Employee_position='somejob'/*+IGNORECONST(2)*;
CREATE DIRECTED QUERY
```
ACTIVATE DIRECTED QUERY findEmployees;
ACTIVATE DIRECTED QUERY

IGNORECONST requires an integer argument. This argument matches constants in input and annotated queries that you want the optimizer to ignore. In the previous example, the input and annotated queries of the directed query, findEmployees, use IGNORECONST to pair two sets of constants:

- IGNORECONST(1) pairs input and annotated query settings for Employee_city.
- IGNORECONST(2) pairs input and annotated query settings for Employee_position.

When the optimizer maps input queries to the directed query findEmployees, the IGNORECONST arguments for Employee_city and Employee_position tell it to ignore the saved values for these two columns. Thus, users can supply any values for these columns.

For example, the following query plan shows how the optimizer maps one user query to the directed query findEmployees:

QUERY PLAN DESCRIPTION:
--------------------------------
EXPLAIN SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension WHERE Employee_city='Boston' AND Employee_position='Cashier';

The following active directed query is being executed:
SELECT EMP_Dimension.Employee_first_name, EMP_Dimension.Employee_last_name FROM public.Emp_Dimension EMP_Dimension/*+projs('public.Emp_Dimension_Unseg')*/ WHERE (EMP_Dimension.Employee_city = 'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7)))

Access Path:
+STORAGE ACCESS for EMP_Dimension [Cost: 154, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
 | Projection: public.Emp_Dimension_Unseg
 | Materialize: EMP_Dimension.Employee_first_name, EMP_Dimension.Employee_last_name
 | Filter: (EMP_Dimension.Employee_city = 'Boston')
 | Filter: (Emp_Dimension.Employee_position = 'Cashier')
 | Execute on: Query Initiator

Embedding IGNORECONST Hints in Optimizer-Generated Directed Queries

You can embed IGNORECONST hints in the input query argument of CREATE DIRECTED QUERY OPTIMIZER. The optimizer creates an annotated query that includes IGNORECONST hints for the corresponding columns. For example, given this CREATE DIRECTED QUERY OPTIMIZER statement:
CREATE DIRECTED QUERY
findGreetersAnyCity
SELECT Employee_first_name, Employee_last_name
FROM EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/
WHERE Employee_city='anywhere'/*+IGNORECONST(9)*/ AND Employee_position='Greeter';

the optimizer creates the following annotated query:

SELECT /*+verbatim */ EMP_Dimension.Employee_first_name AS Employee_first_name, EMP_Dimension.Employee_last_name AS Employee_last_name
FROM public.Emp_Dimension AS EMP_Dimension /*+projs('public.Emp_Dimension_Unseg')*/
WHERE (EMP_Dimension.Employee_city = 'anywhere'::varchar(8) /*+IgnoreConst(9)*/) AND (EMP_Dimension.Employee_position = 'Greeter'::varchar(7))

Mapping One-to-Many IGNORECONST Hints

The examples shown so far demonstrate one-to-one pairings of IGNORECONST hints. You can also use IGNORECONST to map one input constant to multiple constants in an annotated query. This approach can be especially useful when you want to provide the optimizer with explicit instructions how to execute a query that joins tables.

For example, this simple query joins two tables:

SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 8;

In this case, the optimizer can infer that S.a and T.b have the same value and implements the join accordingly. In other cases, you might want to provide the optimizer explicit guidance on implementing a join condition. Given the previous input query, you can create a directed query that uses the IGNORECONST hint to map query input for S.a to S.a and T.b:

=> SAVE QUERY SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 999/*+IGNORECONST(1)*/; SAVE QUERY

=> CREATE DIRECTED QUERY CUSTOM joinST SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 999/*+IGNORECONST(1)*/ AND T.b = 999/*+IGNORECONST(1)*/; CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY joinST; ACTIVATE DIRECTED QUERY

Now, given the following input query:

SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 3;

the optimizer uses the directed query joinST and its ignoreconst hints to rewrite the query as follows:
QUERY PLAN DESCRIPTION:
----------------------------------

EXPLAIN SELECT * FROM S JOIN T ON S.a = T.b WHERE S.a = 3;

The following active directed query (query name: joinST) is being executed:
SELECT S.a, T.b FROM (public.S JOIN public.T ON (S.a = T.b)) WHERE ((S.a = 3) AND (T.b = 3))

Access Path:
+--JOIN MERGEJOIN(inputs presorted) [Cost: 11, Rows: 9 (NO STATISTICS)] (PATH ID: 1)
  | Join Cond: (S.a = T.b)
  | Execute on: v_vmart_node0002
  | +-- Outer -> STORAGE ACCESS for S [Cost: 5, Rows: 9 (NO STATISTICS)] (PATH ID: 2)
  |     | Projection: public.S_b0
  |     | Materialize: S.a
  |     | Filter: (S.a = 3)
  |     | Execute on: v_vmart_node0002
  |     | Runtime Filter: (SIP1(MergeJoin): S.a)
  | +-- Inner -> STORAGE ACCESS for T [Cost: 5, Rows: 9 (NO STATISTICS)] (PATH ID: 3)
  |     | Projection: public.T_b0
  |     | Materialize: T.b
  |     | Filter: (T.b = 3)
  |     | Execute on: v_vmart_node0002

Rewriting Queries

You can use directed queries to change the semantics of a given query—that is, substitute one query for another. This can be especially important when you have little or no control over the content and format of input queries that your Vertica database processes. You can map these queries to directed queries that rewrite the original input for optimal execution.

The following sections describe two use cases:

- **Rewriting Join Queries**
- **Using Query Templates**

Rewriting Join Queries

Many of your input queries join multiple tables. With the recent introduction of flattened tables, you've determined that in many cases, it would be more efficient to denormalize much of your data in several wide tables and query those tables directly. You cannot revise the input queries themselves. However, you can use directed queries to map these queries to queries on the flattened table data.
For example, the following query aggregates regional sales of white wine products, by joining three tables in the VMart database:

```
=> SELECT SD.store_region AS 'Sales Region',
    SD.store_city AS 'City',
    SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=\P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
```

You can consolidate the joined table data in a single flattened table, and query this table instead. By doing so, you can access the same data faster. You can create this table with the following SQL:

```
CREATE TABLE store.store_sales_wide AS SELECT * FROM store.store_sales_fact;
ALTER TABLE store.store_sales_wide ADD COLUMN store_name VARCHAR(64)
SET USING (SELECT store_name FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_city varchar(64)
SET USING (SELECT store_city FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_state char(2)
SET USING (SELECT store_state char FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide ADD COLUMN store_region varchar(64)
SET USING (SELECT store_region FROM store.store_dimension
  WHERE store.store_sales_wide.store_key=store.store_dimension.store_key);
ALTER TABLE store.store_sales_wide add column product_description VARCHAR(128)
SET USING (SELECT product_description FROM public.product_dimension
  WHERE store_sales_wide.product_key||store_sales_wide.product_version = product_dimension.product_key||product_dimension.product_version);
ALTER TABLE store.store_sales_wide ADD COLUMN sku_number char(32)
SET USING (SELECT sku_number char FROM product_dimension
  WHERE store_sales_wide.product_key||store_sales_wide.product_version = product_dimension.product_key||product_dimension.product_version);

SELECT REFRESH_COLUMNS ('store.store_sales_wide','','rebuild');
```

After creating this table and refreshing its SET USING columns, you can rewrite the earlier query as follows:

```
=> SELECT store_region AS 'Sales Region',
    store_city AS 'City',
    SUM(gross_profit_dollar_amount) AS Total
FROM store.store_sales_wide
WHERE product_description ILIKE '%wine%' AND product_description ILIKE '%white%'
GROUP BY ROLLUP (store_region, store_city)
ORDER BY 1,3 DESC;
```

<table>
<thead>
<tr>
<th>Sales Region</th>
<th>City</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td></td>
<td></td>
</tr>
<tr>
<td>East</td>
<td>Sterling Heights</td>
<td>762632</td>
</tr>
<tr>
<td>East</td>
<td></td>
<td>67391</td>
</tr>
</tbody>
</table>
Querying the flattened table is more efficient; however, you still must account for input queries that continue to use the earlier join syntax. You can do so by creating a custom directed query, which redirects these input queries to the desired syntax:

1. **Save the input query:**

```sql
=> SAVE QUERY SELECT SD.store_region AS 'Sales Region',
    SD.store_city AS 'City',
    SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white%'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;
SAVE QUERY
```

2. **Map the saved query to a directed query with the desired syntax, and activate the directed query:**

```sql
=> CREATE DIRECTED QUERY CUSTOM 'RegionalSalesWhiteWine'
SELECT store_region AS 'Sales Region',
    store_city AS 'City',
    SUM(gross_profit_dollar_amount) AS Total
FROM store.store_sales_wide
WHERE product_description ILIKE '%wine%' AND product_description ILIKE '%white%'
```
GROUP BY ROLLUP (store_region, store_city)
ORDER BY 1,3 DESC;
CREATE DIRECTED QUERY

=> ACTIVATE DIRECTED QUERY RegionalSalesWhiteWine;
ACTIVATE DIRECTED QUERY

When directed query RegionalSalesWhiteWine is active, the query optimizer maps all queries that match the original input format to the directed query, as shown in the following query plan:

=> EXPLAIN SELECT SD.store_region AS 'Sales Region',
   SD.store_city AS 'City',
   SUM(SF.gross_profit_dollar_amount) AS Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%wine%' AND P.product_description ILIKE '%white'
GROUP BY ROLLUP (SD.store_region, SD.store_city)
ORDER BY 1,3 DESC;

The following active directed query (query name: RegionalSalesWhiteWine) is being executed:
SELECT store_sales_wide.store_region AS "Sales Region",
store_sales_wide.store_city AS City,
sum(store_sales_wide.gross_profit_dollar_amount) AS Total FROM store.store_sales_wide
WHERE ((store_sales_wide.product_description ~~* '%wine%'::varchar(6))
AND (store_sales_wide.product_description ~~* '%white%'::varchar(7)))
GROUP BY GROUPING SETS((store_sales_wide.store_region, store_sales_wide.store_city), (store_sales_wide.store_region), ()),
ORDER BY store_sales_wide.store_region, sum(store_sales_wide.gross_profit_dollar_amount) DESC

Access Path:
+-SORT [Cost: 76K, Rows: 100] (PATH ID: 1)
 | Order: store_sales_wide.store_region ASC, sum(store_sales_wide.gross_profit_dollar_amount) DESC
 | Execute on: All Nodes
 | ++-- GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 76K, Rows: 100] (PATH ID: 2)
 | | Aggregates: sum(store_sales_wide.gross_profit_dollar_amount)
 | | Group By: store_sales_wide.store_region, store_sales_wide.store_city
 | | Grouping Sets: (store_sales_wide.store_region, store_sales_wide.store_city, <SVAR>),
    (store_sales_wide.store_region, <SVAR>), (<SVAR>)
 | | Execute on: All Nodes
 | | ++-- STORAGE ACCESS for store_sales_wide [Cost: 23K, Rows: 2M] (PATH ID: 3)
 | | | Projection: store_sales_wide.store_region, store_sales_wide.store_city,
    store_sales_wide.store_region
 | | | Filter: ((store_sales_wide.product_description ~~* '%wine%') AND (store_sales_wide.product_description ~~* '%white%'))
 | | | Execute on: All Nodes

To compare the costs of executing the directed query and executing the original input query, deactivate the directed query and use EXPLAIN on the original input query. The optimizer reverts to creating a plan for the input query that incurs significantly greater cost, as shown in the following query plan:
Using Query Templates

You can use directed queries to implement multiple queries that are identical except for the predicate strings on which query results are filtered. For example, directed query `RegionalSalesWhiteWine` only handles input queries that filter on `product_description` values containing the strings `wine` and `white`. You can create a modified
version of this directed query that matches the syntax of multiple input queries, which differ only in their pairs of input values—for example, wine and red.

You create this query template in the following steps:

1. Use the input query on the flattened table to create an **optimized-generated directed query**:

   ```sql
   => CREATE DIRECTED QUERY optimizer RegionalSalesTemp
      SELECT store_region AS 'Sales Region',
              store_city AS 'City',
              SUM(gross_profit_dollar_amount) AS Total
      FROM store.store_salesWide
      WHERE product_description ILIKE '%wine%'
      AND product_description ILIKE '%white%'
      GROUP BY ROLLUP (store_region, store_city)
      ORDER BY 1,3 DESC;
   CREATE DIRECTED QUERY
   ```

2. Modify the original join input query with **IGNORECONST** hints and **save it**:

   ```sql
   => SAVE QUERY SELECT SD.store_region AS 'Sales Region',
                  SD.store_city AS 'City',
                  SUM(SF.gross_profit_dollar_amount) Total
      FROM store.store_sales_fact SF
      JOIN store.store_dimension SD ON SF.store_key=SD.store_key
      JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
      WHERE P.product_description ILIKE 'desc-string1' /*+IGNORECONST(1)*/
      AND P.product_description ILIKE 'desc-string2' /*+IGNORECONST(2)*/
      GROUP BY ROLLUP (SD.store_region, SD.store_city)
      ORDER BY 1,3 DESC;
   SAVE QUERY
   ```

3. **Get the optimizer-generated query from system table** **DIRECTED_QUERIES**:

   ```sql
   => \x
   => SELECT annotated_query FROM DIRECTED_QUERIES WHERE query_name='RegionalSalesTemp';
   [ RECORD 1 ]----------------------------------------------
   annotated_query | SELECT /*verbatim*/ store_sales_wide.store_region AS "Sales Region",
                     store_sales_wide.store_city AS City,
                     sum(store_sales_wide.gross_profit_dollar_amount) AS Total
      FROM store.store_sales_wide AS store_sales_wide/**projs('store.store_salesWide')*/
      WHERE ((store_sales_wide.product_description ~* '%wine':varchar(6))
      AND (store_sales_wide.product_description ~* '%white':varchar(7)))
      GROUP BY /*+GBYType(Hash)*/ GROUPING SETS((1, 2), (1), ())
      ORDER BY 1 ASC, 3 DESC
   (1 row)
   ```

4. Create a modified version of the annotated query that qualifies the input values with **IGNORECONST** hints. Use this version to create a custom directed query, and then activate it:
When following the template.

After activating this directed query, Vertica can use it for input queries that match the template. These queries can use any pair of strings to filter the result set. For example, the following input query filters on the strings chicken and frozen:

```sql
EXPLAIN SELECT SD.store_region AS 'Sales Region',
    SD.store_city AS 'City',
    SUM(SF.gross_profit_dollar_amount) Total
FROM store.store_sales_fact SF
JOIN store.store_dimension SD ON SF.store_key=SD.store_key
JOIN product_dimension P ON SF.product_key||SF.product_version=P.product_key||P.product_version
WHERE P.product_description ILIKE '%chicken%' AND P.product_description ILIKE '%frozen%'
GROUP BY /*+GByType(Hash)*/ GROUPING SETS((store_salesWide.store_region, store_salesWide.store_city), (store_salesWide.store_region), ())
ORDER BY store_salesWide.store_region, sum(store_salesWide.gross_profit_dollar_amount) DESC
```

The following active directed query (query name: RegionalSales) is being executed:

```sql
SELECT /*+verbatim*/ store_salesWide.store_region AS "Sales Region",
    store_salesWide.store_city AS "City",
    sum(store_salesWide.gross_profit_dollar_amount) AS Total
FROM store.store_salesWide AS store_salesWide/*+projs('store.store_salesWide')*/
WHERE ((store_salesWide.product_description ILIKE '%chicken%':varchar(9)))
    AND (store_salesWide.product_description ILIKE '%frozen%':varchar(8))
GROUP BY /*+GByType(Hash)*/ GROUPING SETS((store_salesWide.store_region, store_salesWide.store_city), (store_salesWide.store_region), ())
ORDER BY store_salesWide.store_region, sum(store_salesWide.gross_profit_dollar_amount) DESC
```

Access Path:
+SORT [Cost: 214K, Rows: 100] (PATH ID: 1)
  |  Order: store_salesWide.store_region ASC, sum(store_salesWide.gross_profit_dollar_amount) DESC
  |  Execute on: All Nodes
  |  ----> GROUPBY HASH (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 214K, Rows: 100] (PATH ID: 2)
  |  |  Aggregates: sum(store_salesWide.gross_profit_dollar_amount)
  |  |  Group By: store_salesWide.store_region, store_salesWide.store_city
  |  |  Grouping Sets: (store_salesWide.store_region, store_salesWide.store_city, <SVAR>), (store_salesWide.store_region, <SVAR>), (<SVAR>)
  |  |  Execute on: All Nodes
  |  |  ----> STORAGE ACCESS for store_salesWide [Cost: 39K, Rows: 5M] (PATH ID: 3)
  |  |  |  Projection: store.salesWide.super
  |  |  |  |  Materialize: store_salesWide.gross_profit_dollar_amount, store_salesWide.store_city, store_salesWide.store_region
  |  |  |  Filter: ((store_salesWide.product_description ILIKE '%chicken%') AND (store_salesWide.product_description ILIKE '%frozen%'))
  |  |  Execute on: All Nodes

When you execute this query, it returns with the following results:
### Managing Directed Queries

Vertica provides a number of ways to manage directed queries:

- **Getting Directed Queries**
- **Identifying Active Directed Queries**
- **Activating and Deactivating Directed Queries**
- **Exporting Directed Queries from the Catalog**
- **Dropping Directed Queries**

### Getting Directed Queries

You can obtain catalog information about directed queries in two ways:
- Run the statement `GET DIRECTED QUERY`.
- Query the system table `V_CATALOG.DIRECTED_QUERIES`.

### Run GET DIRECTED QUERY

`GET DIRECTED QUERY` queries the system table `V_CATALOG.DIRECTED_QUERIES` on the specified input query. It returns a list of directed queries that map to the input query.

The following `GET DIRECTED QUERY` statement returns two directed queries that map to the same input query, `findBostonCashiers_OPT` and `findBostonCashiers_CUSTOM`:

```sql
=> GET DIRECTED QUERY
SELECT employee_first_name, employee_last_name
    FROM employee_dimension WHERE employee_city='Boston' AND job_title='Cashier';

- [ RECORD 1 ]----+
  query_name | findBostonCashiers_OPT
  is_active   | f
  vertica_version | Vertica Analytic Database v7.2.3
  comment     | Optimizer-generated directed query
  creation_date | 2016-04-25 08:17:26.913339
  annotated_query | SELECT /*+ verbatim */ employee_dimension.employee_first_name AS employee_first_name,
                   employee_dimension.employee_last_name AS employee_last_name
                   FROM public.employee_dimension AS employee_dimension/*+projs('public.employee_dimension')*/
                   WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7))
- [ RECORD 2 ]----+
  query_name | findBostonCashiers_CUSTOM
  is_active   | t
  vertica_version | Vertica Analytic Database v7.2.3
  comment     | Custom directed query
  creation_date | 2016-04-25 09:15:11.464417
  annotated_query | SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
                   FROM public.employee_dimension/*+projs('public.Emp_Dimension_Unseg')*/
                   WHERE ((employee_dimension.employee_city = 'Boston':varchar(6)) AND (employee_dimension.job_title = 'Cashier':varchar(7)))
```

### Query V_CATALOG.DIRECTED_QUERIES

You can query the system table `V_CATALOG.DIRECTED_QUERIES` directly. For example:

```sql
=> SELECT query_name, is_active FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name ILIKE '%findBostonCashiers%';

-----------------------------------------
  query_name | is_active
-----------------------------------------
  findBostonCashiers_CUSTOM | t
  findBostonCashiers_OPT    | f
(2 rows)
```

(2 rows)
Caution: Query results for the fields input_query and annotated_query are truncated after 8192 characters. You can get the full content of both fields in two ways:

- Use the statement `GET DIRECTED QUERY`.
- Use `EXPORT_CATALOG` to export directed queries.

Identifying Active Directed Queries

Multiple directed queries can map to the same input query. The `is_active` column returned by `GET DIRECTED QUERY` clarifies which directed queries are active. If multiple directed queries are active for the same input query, the optimizer uses the first one to be created. In that case, you can use `EXPLAIN` to identify which directed query is active.

Note: It is good practice to activate only one directed query at a time for a given input query.

In the following example, `GET DIRECTED QUERY` returns with two directed queries that map to the same input query: `findBostonCashiers_OPT`, and `findBostonCashiers_CUSTOM`. Only `findBostonCashiers_CUSTOM` is flagged as active:

```sql
=> GET DIRECTED QUERY SELECT Employee_first_name, Employee_last_name FROM Emp_Dimension
   WHERE Employee_city='Boston' and Employee_position='Cashier';
- [ RECORD 1 ] ++-
 query_name     | findBostonCashiers_OPT
 is_active      | f
 vertica_version| Vertica Analytic Database v7.2.0
 comment        | Optimizer-generated directed query
 creation_date  | 2015-09-02 09:36:29.395702
 annotated_query| SELECT /*+ verbatim */ Emp_Dimension.Employee_first_name AS Employee_first_name,
              | Emp_Dimension.Employee_last_name AS Employee_last_name
              | FROM public.Emp_Dimension AS Emp_Dimension/*+projs('public.Emp_Dimension')*/
              | WHERE (Emp_Dimension.Employee_city = 'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7))
- [ RECORD 2 ] ++-
 query_name     | findBostonCashiers_CUSTOM
 is_active      | t
 vertica_version| Vertica Analytic Database v7.2.0-20150902
 comment        | Custom directed query
 creation_date  | 2015-09-02 13:27:38.225568
 annotated_query| SELECT Emp_Dimension.Employee_first_name, Emp_Dimension.Employee_last_name FROM public.Emp_Dimension/*+projs('public.Emp_Dimension_Unseg')*/ WHERE ((Emp_Dimension.Employee_city = 'Boston'::varchar(6)) AND (Emp_Dimension.Employee_position = 'Cashier'::varchar(7)))
```

If you run `EXPLAIN` on the same input query, it returns with a query plan that confirms use of `findBostonCashiers_CUSTOM` as the active directed query:
Activating and Deactivating Directed Queries

The optimizer uses only directed queries that are active. If multiple directed queries share the same input query, the optimizer uses the first one to be created.

You activate and deactivate directed queries with `ACTIVATE DIRECTED QUERY` and `DEACTIVATE DIRECTED QUERY`, respectively. For example, the following `ACTIVATE DIRECTED QUERY` statement deactivates `findBostonCashiers_OPT` and activates `findBostonCashiers_CUSTOM`:

```sql
=> DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
DEACTIVATE DIRECTED QUERY;
=> ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
ACTIVATE DIRECTED QUERY;
```

Vertica uses the active directed query for a given query across all sessions until it is explicitly deactivated by `DEACTIVATE DIRECTED QUERY` or removed from storage by `DROP DIRECTED QUERY`. If a directed query is active at the time of database shutdown, Vertica automatically reactivates it when you restart the database.

After a direct query is deactivated, the query optimizer handles subsequent invocations of the input query by using another directed query, if one is available. Otherwise, it generates its own query plan.
Exporting Directed Queries from the Catalog

Tip: You can also export query plans as directed queries to an external SQL file. See Batch Query Plan Export.

Before upgrading to a new version of Vertica, you can export directed queries for those queries whose optimal performance is critical to your system:

1. Use EXPORT_CATALOG with the argument DIRECTED_QUERIES to export from the database catalog all current directed queries and their current activation status:

   ```sql
   => SELECT EXPORT_CATALOG('./export_directedqueries', 'DIRECTED_QUERIES');
   EXPORT_CATALOG
   Catalog data exported successfully
   ```

2. EXPORT_CATALOG creates a script to recreate the directed queries, as in the following example:

   ```sql
   SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
   FROM public.employee_dimension WHERE ((employee_dimension.employee_city = 'Boston'::varchar(6))
   AND (employee_dimension.job_title = 'Cashier'::varchar(7)));
   CREATE DIRECTED QUERY CUSTOM findBostonCashiers_OPT COMMENT 'Optimizer-generated directed query'
   OPTVER 'Vertica Analytic Database v7.2.3-20160425' PSDATE '2016-04-25 08:17:26.913339' SELECT
   /*+ verbatim */ employee_dimension.employee_first_name AS employee_first_name,
   employee_dimension.employee_last_name AS employee_last_name
   FROM public.employee_dimension AS employee_dimension/*+projs('public.Emp_Dimension_Unseg')*/
   WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title = 'Cashier'::varchar(7));
   DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;
   
   SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
   FROM public.employee_dimension WHERE ((employee_dimension.employee_city = 'Boston'::varchar(6))
   AND (employee_dimension.job_title = 'Cashier'::varchar(7)));
   CREATE DIRECTED QUERY CUSTOM findBostonCashiers_CUSTOM COMMENT 'Custom directed query'
   OPTVER 'Vertica Analytic Database v7.2.3-20160425' PSDATE '2016-04-25 09:15:11.464417' SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
   FROM public.employee_dimension/*+Projs('public.Emp_Dimension_Unseg')*/
   WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) AND (employee_dimension.job_title = 'Cashier'::varchar(7));
   ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
   ```

Note: The script that EXPORT_CATALOG creates specifies to recreate all directed queries with CREATE DIRECTED QUERY CUSTOM, regardless of how they were created originally.
3. After the upgrade is complete, remove each directed query from the database catalog with `DROP DIRECTED QUERY`. Alternatively, edit the export script and insert a `DROP DIRECTED QUERY` statement before each `CREATE DIRECTED QUERY` statement. For example, you might modify the script generated earlier with the changes shown in bold:

```sql
SAVE QUERY SELECT employee_dimension.employee_first_name, ...
DROP DIRECTED QUERY findBostonCashiers_OPT
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_OPT COMMENT 'Optimizer-generated ...
DEACTIVATE DIRECTED QUERY findBostonCashiers_OPT;

SAVE QUERY SELECT employee_dimension.employee_first_name, ...
DROP DIRECTED QUERY findBostonCashiers_CUSTOM
CREATE DIRECTED QUERY CUSTOM findBostonCashiers_CUSTOM COMMENT 'Custom directed query'...
ACTIVATE DIRECTED QUERY findBostonCashiers_CUSTOM;
```

4. When you run this script, Vertica recreates the directed queries and restores their activation status:

```sql
=> \i /home/dbadmin/export_directedqueries
SAVE QUERY
DROP DIRECTED QUERY
CREATE DIRECTED QUERY
DEACTIVATE DIRECTED QUERY
SAVE QUERY
DROP DIRECTED QUERY
CREATE DIRECTED QUERY
DEACTIVATE DIRECTED QUERY
```

### Dropping Directed Queries

`DROP DIRECTED QUERY` removes the specified directed query from the database catalog. If the directed query is active, Vertica deactivates it before removal.

For example:

```sql
=> DROP DIRECTED QUERY findBostonCashiers_CUSTOM;
DROP DIRECTED QUERY
```

### Batch Query Plan Export

Before upgrading to a new Vertica version, you might wish to use directed queries to save query plans for possible reuse in the new database. You cannot predict which query plans are likely candidates for reuse, so you probably want to save query plans for many, or all, database queries. However, you run hundreds of queries each day. Saving query plans for each one to
the database catalog through repetitive calls to `CREATE DIRECTED QUERY` is impractical. Moreover, doing so can significantly increase catalog size and possibly impact performance. In this case, you can bypass the database catalog and batch export query plans as directed queries to an external SQL file. By offloading query plan storage, you can save any number of query plans from the current database without impacting catalog size and performance. After the upgrade, you can decide which query plans you wish to retain in the new database, and selectively import the corresponding directed queries.

Vertica provides a set of meta-functions that support this approach:

- **EXPORT_DIRECTED_QUERIES** generates query plans from a set of input queries, and writes SQL for creating directed queries that encapsulate those plans.

- **IMPORT_DIRECTED_QUERIES** imports to the database catalog directed queries from a SQL file that was generated by `EXPORT_DIRECTED_QUERIES`.

### Exporting Directed Queries

You can batch export any number of query plans as directed queries to an external SQL file, as follows:

1. Create a SQL file that contains the input queries whose query plans you wish to save. See [Output File](#) below.

2. Call the meta-function `EXPORT_DIRECTED_QUERIES` on that SQL file. The meta-function takes two arguments:
   - The input queries file.
   - The name of an external file. `EXPORT_DIRECTED_QUERIES` writes SQL for creating directed queries to this file. If you supply an empty string, Vertica writes the SQL to standard output. For details, see [Output File](#) below.

For example, the following `EXPORT_DIRECTED_QUERIES` statement specifies input file `inputQueries` and output file `outputQueries`:

```sql
=> SELECT EXPORT_DIRECTED_QUERIES('/home/dbadmin/inputQueries','/home/dbadmin/outputQueries');
EXP...DIRECTED_QUERIES
1 queries successfully exported.
Queries exported to /home/dbadmin/outputQueries.
```

(1 row)
Input File

The input file that you supply to EXPORT_DIRECTED_QUERIES contains one or more input queries. For each input query, you can optionally specify two fields that are used in the generated directed query:

- DirQueryName provides the directed query's unique identifier, a string that conforms to conventions described in Identifiers.

- DirQueryComment specifies a quote-delimited string, up to 128 characters.

You format each input query as follows:

```plaintext
--DirQueryName=query-name
--DirQueryComment='comment'
input-query
```

For example, a file can specify one input query as follows:

```plaintext
--DirQueryName=FindEmployeesBoston
--DirQueryComment='This query finds all Boston employees, ordered by position'
SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name, employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
```

Output File

EXPORT_DIRECTED_QUERIES generates SQL for creating directed queries, and writes the SQL to the specified file or to standard output. In both cases, output conforms to the following format:

```plaintext
/* Query: directed-query-name */
/* Comment: directed-query-comment */
SAVE QUERY input-query;
CREATE DIRECTED QUERY CUSTOM 'directed-query-name'
COMMENT 'directed-query-comment'
OPTVER 'vertica-release-num'
PSDATE 'timestamp'
annotated-query
```

For example, given the previous input, Vertica writes the following output to /home/dbadmin/outputQueries:

```plaintext
/* Query: FindEmployeesBoston */
/* Comment: This query finds all Boston employees, ordered by position */
SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name, employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
```
'Boston':varchar(6)) ORDER BY employee_dimension.job_title;
CREATE DIRECTED QUERY CUSTOM 'FindEmployeesBoston'
COMMENT 'This query finds all Boston employees, ordered by position'
OPTVER 'Vertica Analytic Database v8.0.1-20161013'
PSDATE '2016-10-13 08:59:58.054505'
SELECT /*+verbatim*/employee_dimension.employee_first_name AS employee_first_name,
employee_dimension.employee_last_name AS employee_last_name,
employee_dimension.job_title AS job_title
FROM public.employee_dimension
WHERE employee_dimension.employee_city = 'Boston':varchar(6))
ORDER BY 3 ASC;

If a given input query omits DirQueryName and DirQueryComment fields, EXPORT_DIRECTED_QUERIES automatically generates the following output:

* Query: Autoname:timestamp.n */
  where n is a zero-based integer index that ensures uniqueness among auto-generated names with the same timestamp.

* Comment: Optimizer-generated directed query */

For example, the following input file contains one SELECT statement, and omits the DirQueryName and DirQueryComment fields:

SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name
FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston':varchar(6))
ORDER BY employee_dimension.job_title

Given this file, EXPORT_DIRECTED_QUERIES generates the following output:

/* Query: Autoname:2016-10-13 09:44:33.527548.0 */
/* Comment: Optimizer-generated directed query */
SAVE QUERY SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name,
employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) ORDER BY employee_dimension.job_title;
CREATE DIRECTED QUERY CUSTOM 'Autoname:2016-10-13 09:44:33.527548.0'
COMMENT 'Optimizer-generated directed query'
...

Error File

If any errors or warnings occur during EXPORT_DIRECTED_QUERIES execution, it returns with a message like this one:

1 queries successfully exported.
1 warning message was generated.
Queries exported to /home/dbadmin/outputQueries.
See error report, /home/dbadmin/outputQueries.err for details.

EXPORT_DIRECTED_QUERIES writes all errors and warnings to a file that it creates on the same path as the output file, and uses the output file's base name.
In the previous example, the output filename is `/home/dbadmin/outputQueries`, so `EXPORT_DIRECTED_QUERIES` writes errors to `/home/dbadmin/outputQueries.err`.

The error file can capture a number of errors, such as all instances where `EXPORT_DIRECTED_QUERIES` was unable to create a directed query. In the following example, the error file contains a warning that no name field was supplied for the specified input query, and records the name that was auto-generated for it:

```
WARNING: Name field not supplied. Using auto-generated name: 'Autoname:2016-10-13 09:44:33.527548.0'
Input Query: SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name, employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston'::varchar(6)) ORDER BY employee_dimension.job_title;
END WARNING
```

### Importing Directed Queries

After you determine which exported query plans you wish to use in the current database, you import them with `IMPORT_DIRECTED_QUERIES`. You supply this function with the name of the export file that you created with `EXPORT_DIRECTED_QUERIES`, and the names of directed queries you wish to import. For example:

```
=> SELECT IMPORT_DIRECTED_QUERIES('/home/dbadmin/outputQueries','FindEmployeesBoston');

1 directed queries successfully imported.
To activate a query named 'my_query1':
=>ACTIVATE DIRECTED QUERY 'my_query1';

(1 row)
```

After importing the desired directed queries, you must activate them with `ACTIVATE DIRECTED QUERY` before you can use them to create query plans.

### Half Join and Cross Join Semantics

The Vertica optimizer uses several keywords in directed queries to recreate cross join and half join subqueries. It also supports an additional set of keywords to express complex cross joins and half joins. You can also use these keywords in queries that you execute directly in vsql.

**Caution:** These keywords do not conform with standard SQL; they are intended for use only by the Vertica optimizer.

For details, see the following topics:
Half-Join Subquery Semantics

The Vertica optimizer uses several keywords in directed queries to recreate half-join subqueries with certain search operators, such as ANY or NOT IN:

- **SEMI JOIN**
- **NULLAWARE ANTI JOIN**
- **SEMIALL JOIN**
- **ANTI JOIN**

SEMI JOIN

Recreates a query that contains a subquery preceded by an IN, EXIST, or ANY operator and executes a semi-join.

**Input query**

```
SELECT product_description FROM product_dimension
WHERE product_dimension.product_key IN (SELECT qty_in_stock from inventory_fact);
```

**Query plan**

```
explain SELECT product_description FROM product_dimension WHERE product_dimension.product_key IN (SELECT qty_in_stock from inventory_fact);

Access Path:
+-JOIN HASH [Semi] [Cost: 1K, Rows: 30K] (PATH ID: 1) Outer (FILTER) Inner (RESEGMENT)
  | Join Cond: (product_dimension.product_key = VAL(2))
  | Materialize at Output: product_dimension.product_description
  | Execute on: All Nodes
  | +-- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
  |     | Projection: public.product_dimension
  |     | Materialize: product_dimension.product_key
  |     | Execute on: All Nodes
  |     | Runtime Filter: (SIP1(HashJoin): product_dimension.product_key)
  |     | +-- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
  |     |     | Execute on: All Nodes
  |     |     | +--- STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
```
Optimizer-generated annotated query

```
SELECT /*+ syntactic_join */ product_dimension.product_description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
SEMI JOIN /*+Distrib(F,R),JType(H)*/ (SELECT inventory_fact.qty_in_stock
AS qty_in_stock
FROM public.inventory_fact
AS inventory_fact/*+projs('public.inventory_fact')*/)
AS subQ_1
ON (product_dimension.product_key = subQ_1.qty_in_stock))
```

NULLAWARE ANTI JOIN

Recreates a query that contains a subquery preceded by a NOT IN or !=ALL operator, and executes a null-aware anti-join.

Input query

```
SELECT product_description FROM product_dimension
WHERE product_dimension.product_key NOT IN (SELECT qty_in_stock from inventory_fact);
```

Query plan

```
EXPLAIN SELECT product_description FROM product_dimension WHERE product_dimension.product_key not IN (SELECT qty_in_stock from inventory_fact);
```

Optimizer-generated annotated query

```
| | | Projection: public.inventory_fact_b0
| | | Materialize: inventory_fact.qty_in_stock
| | | Execute on: All Nodes

| | Projection: public.product_dimension_880_dbd_2_rep_VMartDesign
| | Materialize: product_dimension.product_key
| | Execute on: Query Initiator
| | | | Outer --> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
| | | | | Projection: public.product_dimension_880_dbd_2_rep_VMartDesign
| | | | | Materialize: product_dimension.product_key
| | | | | Execute on: Query Initiator
| | | | Inner --> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
| | | | | Execute on: All Nodes
| | | | | | STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
| | | | | | | Projection: public.inventory_fact_880_dbd_9_seg_VMartDesign_b0
| | | | | | | Materialize: inventory_fact.qty_in_stock
| | | | | | | Execute on: All Nodes
```
SEMIALL JOIN

Recreates a query that contains a subquery preceded by an ALL operator, and executes a semi-all join.

Input query

```sql
SELECT product_key, product_description FROM product_dimension
WHERE product_dimension.product_key > ALL (SELECT product_key from inventory_fact);
```

Query plan

```
QUERY PLAN DESCRIPTION:
--------------------
explain SELECT product_key, product_description FROM product_dimension WHERE product_dimension.product_key > ALL (SELECT product_key from inventory_fact);

Access Path:
++ JOIN HASH [Semi][All] [Cost: 7M, Rows: 30K] (PATH ID: 1) Outer (FILTER) Inner (BROADCAST)
  | Join Filter: (product_dimension.product_key > VAL(2))
  | Materialize at Output: product_dimension.product_description
  | Execute on: All Nodes
  | ++ Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
  | | Projection: public.product_dimension
  | | Materialize: product_dimension.product_key
  | | Execute on: All Nodes
  | ++ Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
  | | Execute on: All Nodes
  | +++ STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
  | | | Projection: public.inventory_fact.product_key
  | | | Materialize: inventory_fact.product_key
  | | | Execute on: All Nodes
```

Optimizer-generated annotated query

```sql
SELECT /*+ syntactic_join */ product_dimension.product_key AS product_key, product_dimension.product_description AS product_description FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
SEMIALL JOIN /*+Distrib(F,B),JType(H)*/ (SELECT inventory_fact.product_key AS product_key FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1
ON (product_dimension.product_key > subQ_1.product_key))
```
ANTI JOIN

Recreates a query that contains a subquery preceded by a `NOT EXISTS` operator, and executes an anti-join.

Input query

```sql
SELECT product_key, product_description FROM product_dimension
WHERE NOT EXISTS (SELECT inventory_fact.product_key FROM inventory_fact
WHERE inventory_fact.product_key = product_dimension.product_key);
```

Query plan

```sql
QUERY PLAN DESCRIPTION:

explain SELECT product_key, product_description FROM product_dimension WHERE NOT EXISTS (SELECT inventory_fact.product_key FROM inventory_fact WHERE inventory_fact.product_key = product_dimension.product_key);

Access Path:
+-JOIN HASH [Anti] [Cost: 703, Rows: 30K] (PATH ID: 1) Outer (FILTER)
  | Join Cond: (VAL(1) = product_dimension.product_key)
  | Materialize at Output: product_dimension.product_description
  | Execute on: All Nodes
  |--- Outer -> STORAGE ACCESS for product_dimension [Cost: 152, Rows: 60K] (PATH ID: 2)
  |   | Projection: public.product_dimension_DBD_2_rep_VMartDesign
  |   | Materialize: product_dimension.product_key
  |   | Execute on: All Nodes
  |--- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
  |   | Execute on: All Nodes
  |   |---- STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
  |   |---- Projection: public.inventory_fact_DBD_9_seg_VMartDesign_b0
  |   |--- Materialize: inventory_fact.product_key
  |   | Execute on: All Nodes
```

Optimizer-generated annotated query

```sql
SELECT /*+ syntactic_join */ product_dimension.product_key AS product_key, product_dimension.product_description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/
ANTI JOIN /*+Distrib(F,L),JType(H)*/ (SELECT inventory_fact.product_key AS "inventory_fact.product_key"
FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/ AS subQ_1
ON (subQ_1."inventory_fact.product_key" = product_dimension.product_key))
```

Complex Join Semantics

The Vertica optimizer uses a set of keywords to express complex cross joins and half joins. All complex joins are indicated by the keyword COMPLEX, which is inserted before the keyword
JOIN—for example, CROSS  COMPLEX  JOIN. Semantics for complex half joins have an additional requirement, which is detailed below.

Complex Cross Join

Vertica uses the keyword phrase CROSS  COMPLEX  JOIN to describe all complex cross joins. For example:

Input query

```sql
SELECT
  (SELECT max(sales_quantity) FROM store.store_sales_fact) *
  (SELECT max(sales_quantity) FROM online_sales.online_sales_fact);
```

Query plan

```sql
EXPLAIN SELECT
  (SELECT max(sales_quantity) FROM store.store_sales_fact) *
  (SELECT max(sales_quantity) FROM online_sales.online_sales_fact);

Access Path:
+-JOIN (CROSS JOIN) [Cost: 4K, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
  | Execute on: Query Initiator
  | +- Outer -> JOIN (CROSS JOIN) [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 2)
  |   | Execute on: Query Initiator
  |   | +- Outer -> STORAGE ACCESS for dual [Cost: 1K, Rows: 1] (PATH ID: 3)
  |   |   | Projection: v_catalog.dual_p
  |   |   | Materialize: dual.dummy
  |   | Execute on: Query Initiator
  |   | +- Inner -> SELECT [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 4)
  |   |   | Execute on: Query Initiator
  |   |   | +--> GROUPBY NOTHING [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 5)
  |   |   |   | Aggregates: max(store_sales_fact.sales_quantity)
  |   |   |   | Execute on: All Nodes
  |   |   | +--> STORAGE ACCESS for store_sales_fact [Cost: 1K, Rows: 5M (NO STATISTICS)] (PATH ID: 6)
  |   |   |   | Projection: store.store_sales_fact_super
  |   |   |   | Materialize: store_sales_fact.sales_quantity
  |   |   | Execute on: All Nodes
  |   | +- Inner -> SELECT [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 7)
  |   |   | Execute on: Query Initiator
  |   |   | +--> GROUPBY NOTHING [Cost: 2K, Rows: 1 (NO STATISTICS)] (PATH ID: 8)
  |   |   | Aggregates: max(online_sales_fact.sales_quantity)
  |   |   | Execute on: All Nodes
  |   |   | +--> STORAGE ACCESS for online_sales_fact [Cost: 1K, Rows: 5M (NO STATISTICS)] (PATH ID: 9)
  |   |   | Projection: online_sales.online_sales_fact_super
  |   |   | Materialize: online_sales_fact.sales_quantity
  |   | Execute on: All Nodes
```

Optimizer-generated annotated query
The following annotated query returns the same results as the input query shown earlier. As with all optimizer-generated annotated queries, you can execute this query directly in vsql, either as written or with modifications:

```sql
SELECT /*+syntactic_join,verbatim*/ (subQ_1.max * subQ_2.max) AS "?column?"
FROM ((v_catalog.dual AS dual CROSS COMPLEX JOIN /*+Distrib(L,L),JType(H)*/
(SELECT max(store_sales_fact.sales_quantity) AS max
FROM store.store_sales_fact AS store_sales_fact/*+projs('store.store_sales_fact')*/))
AS subQ_1)
CROSS COMPLEX JOIN /*+Distrib(L,L),JType(H)*/
(SELECT max(online_sales_fact.sales_quantity) AS max
FROM online_sales.online_sales_fact AS online_sales_fact/*+projs('online_sales.online_sales_fact')*/)
AS subQ_2)
```

Complex Half Join

Complex half joins are expressed by one of the following keywords:

- **SEMI COMPLEX JOIN**
- **NULLAWARE ANTI COMPLEX JOIN**
- **SEMIALL COMPLEX JOIN**
- **ANTI COMPLEX JOIN**

An additional requirement applies to all complex half joins: each subquery's SELECT list ends with a dummy column (labeled as `false`) that invokes the Vertica meta-function `complex_join_marker()`. As the subquery processes each row, `complex_join_marker()` returns true or false to indicate the row's inclusion or exclusion from the result set. The result set returns with this flag to the outer query, which can use the flag from this and other subqueries to filter its own result set.

For example, the query optimizer rewrites the following input query as a NULLAWARE ANTI COMPLEX JOIN. The join returns all rows from the subquery with their `complex_join_marker()` flag set to the appropriate Boolean value.

**Input query**

```sql
SELECT product_dimension.product_description FROM public.product_dimension
WHERE (NOT (product_dimension.product_key NOT IN (SELECT inventory_fact.qty_in_stock FROM public.inventory_fact)));
```

**Query plan**

```sql
EXPLAIN SELECT product_dimension.product_description FROM public.product_dimension
WHERE (NOT (product_dimension.product_key NOT IN (SELECT inventory_fact.qty_in_stock FROM
```
public.inventory_fact));

Access Path:
+--- Outer -> STORAGE ACCESS for product_dimension [Cost: 3K, Rows: 30K] (PATH ID: 1) Inner (BROADCAST)
  | Join Cond: (product_dimension.product_key = VAL(2))
  | Materialize at Output: product_dimension.product_description
  | Filter: (NOT VAL(2))
  | Execute on: All Nodes
| +--- Inner -> SELECT [Cost: 248, Rows: 300K] (PATH ID: 3)
  | | Execute on: All Nodes
| +--- Outer -> STORAGE ACCESS for inventory_fact [Cost: 248, Rows: 300K] (PATH ID: 4)
  | | | Execute on: All Nodes

Optimizer-generated annotated query

The following annotated query returns the same results as the input query shown earlier. As with all optimizer-generated annotated queries, you can execute this query directly in vsql, either as written or with modifications. For example, you can control the outer query's output by modifying how its predicate evaluates the flag subQ_1."false".

```
SELECT /*+syntactic_join,verbatim*/ product_dimension.product_description AS product_description
FROM (public.product_dimension AS product_dimension/*+projs('public.product_dimension')*/)
NULLAWARE ANTI COMPLEX JOIN /*+Distrib(L,B),JType(H)*/
(SELECT inventory_fact.qty_in_stock AS qty_in_stock, complex_join_marker() AS "false"
  FROM public.inventory_fact AS inventory_fact/*+projs('public.inventory_fact')*/) AS subQ_1
ON (product_dimension.product_key = subQ_1.qty_in_stock) WHERE (NOT subQ_1."false")
```

Restrictions

Directed queries support a wide range of queries; however, a number of exceptions apply. Vertica handles all exceptions through optimizer-generated warnings. The sections below divide these restrictions into several categories.

Tables and Projections

The following restrictions apply:

- Optimizer-generated directed queries do not support queries that reference system tables or Data Collector tables. One exception applies: explicit and implicit references to `V_CATALOG.DUAL`. 

• Optimizer-generated directed queries do not support queries that include tables with access policies.

• Directed queries do not support tables without projections.

Functions

Queries are not supported that include the following functions:

• Vertica meta-functions

• Pattern-matching functions

• GROUPING_ID with no arguments

Operators and Clauses

Queries are not supported that include the following:

• WITH clauses when materialization is enabled.

• Queries that include date/time literals that reference the current time, such as NOW or YESTERDAY

Data Types

Queries are not supported that include GEOMETRY data types.
Using Text Search

Text search allows you to quickly search the contents of a single CHAR, VARCHAR, LONG VARCHAR, VARBINARY, or LONG VARBINARY field within a table to locate a specific keyword.

You can use this feature on columns that are queried repeatedly regarding their contents. After you create the text index, DML operations become slightly slower on the source table. This performance change results from syncing the text index and source table. Any time an operation is performed on the source table, the text index updates in the background. Regular queries on the source table are not affected.

The text index contains all of the words from the source table's text field and any other additional columns you included during index creation. Additional columns are not indexed— their values are just passed through to the text index. The text index is like any other Vertica table, except it is linked to the source table internally.

First, create a text index on the table you plan to search. Then, after you have indexed your table, run a query against the text index for a specific keyword. This query returns a doc_id for each instance of the keyword. After querying the text index, joining the text index back to the source table should give a significant performance improvement over directly querying the source table about the contents of its text field.

Important: Do not alter the contents or definitions of the text index. If you alter the contents or definitions of the text index, the results do not appropriately match the source table.

Creating a Text Index

In the following example, you perform a text search using a source table called t_log. This source table has two columns:

- One column containing the table's primary key
- Another column containing log file information

You must associate a projection with the source table. Use a projection that is sorted by the primary key and either segmented by hash(id) or unsegmented. You can define this projection on the source table, along with any other existing projections.

Create a text index on the table for which you want to perform a text search.
CREATE TEXT INDEX text_index ON t_log (id, text);

The text index contains two columns:

- `doc_id` uses the unique identifier from the source table.
- `token` is populated with text strings from the designated column from the source table. The word column results from tokenizing and stemming the words found in the text column.

If your table is partitioned then your text index also contains a third column named `partition`.

<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
<th>partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;info&gt;</td>
<td>6</td>
<td>2014</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>2</td>
<td>2014</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>3</td>
<td>2014</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>4</td>
<td>2014</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>5</td>
<td>2014</td>
</tr>
<tr>
<td>database</td>
<td>6</td>
<td>2014</td>
</tr>
<tr>
<td>execute:</td>
<td>6</td>
<td>2014</td>
</tr>
<tr>
<td>object</td>
<td>4</td>
<td>2014</td>
</tr>
<tr>
<td>object</td>
<td>5</td>
<td>2014</td>
</tr>
<tr>
<td>[catalog]</td>
<td>4</td>
<td>2014</td>
</tr>
<tr>
<td>[catalog]</td>
<td>5</td>
<td>2014</td>
</tr>
</tbody>
</table>

You create a text index on a source table only once. In the future, you do not have to re-create the text index each time the source table is updated or changed.

Your text index stays synchronized to the contents of the source table through any operation that is run on the source table. These operations include, but are not limited to:

- COPY
- INSERT
- UPDATE
- DELETE
- DROP PARTITION
- MOVE_PARTITION_TO_TABLE

When you move or swap partitions in a source table that is indexed, verify that the destination table already exists and is indexed in the same way.
Creating a Text Index on a Flex Table

In the following example, you create a text index on a flex table. The example assumes that you have created a flex table called mountains. See Getting Started in Using Flex Tables to create the flex table used in this example.

Before you can create a text index on your flex table, add a primary key constraint to the flex table.

```sql
=> ALTER TABLE mountains ADD PRIMARY KEY (__identity__);
```

Create a text index on the table for which you want to perform a text search. Tokenize the __raw__ column with the FlexTokenizer and specify the data type as LONG VARBINARY. It is important to use the FlexTokenizer when creating text indices on flex tables because the data type of the __raw__ column differs from the default StringTokenizer.

```sql
=> CREATE TEXT INDEX flex_text_index ON mountains(__identity__, __raw__) TOKENIZER public.FlexTokenizer(long varbinary);
```

The text index contains two columns:

- `doc_id` uses the unique identifier from the source table.
- `token` is populated with text strings from the designated column from the source table. The word column results from tokenizing and stemming the words found in the text column.

If your table is partitioned then your text index also contains a third column named `partition`.

```sql
=> SELECT * FROM flex_text_index;

<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.6</td>
<td>5</td>
</tr>
<tr>
<td>Mt</td>
<td>5</td>
</tr>
<tr>
<td>Washington</td>
<td>5</td>
</tr>
<tr>
<td>mountain</td>
<td>5</td>
</tr>
<tr>
<td>12.2</td>
<td>3</td>
</tr>
<tr>
<td>15.4</td>
<td>2</td>
</tr>
<tr>
<td>17000</td>
<td>3</td>
</tr>
<tr>
<td>29029</td>
<td>2</td>
</tr>
<tr>
<td>Denali</td>
<td>3</td>
</tr>
<tr>
<td>Helen</td>
<td>2</td>
</tr>
<tr>
<td>Mt</td>
<td>2</td>
</tr>
<tr>
<td>St</td>
<td>2</td>
</tr>
<tr>
<td>mountain</td>
<td>3</td>
</tr>
<tr>
<td>volcano</td>
<td>2</td>
</tr>
<tr>
<td>29029</td>
<td>1</td>
</tr>
<tr>
<td>34.1</td>
<td>1</td>
</tr>
<tr>
<td>Everest</td>
<td>1</td>
</tr>
</tbody>
</table>
```
You create a text index on a source table only once. In the future, you do not have to re-create the text index each time the source table is updated or changed.

Your text index stays synchronized to the contents of the source table through any operation that is run on the source table. These operations include, but are not limited to:

- COPY
- INSERT
- UPDATE
- DELETE
- DROP PARTITION
- MOVE_PARTITION_TO_TABLE
  When you move or swap partitions in a source table that is indexed, verify that the destination table already exists and is indexed in the same way.

## Searching a Text Index

After you create a text index, write a query to run against the index to search for a specific keyword.

In the following example, you use a WHERE clause to search for the keyword `<WARNING>` in the text index. The WHERE clause should use the stemmer you used to create the text index. When you use the STEMMER keyword, it stems the keyword to match the keywords in your text index. If you did not use the STEMMER keyword, then the default stemmer is `v_txtindex.StemmerCaseInsensitive`. If you used STEMMER NONE, then you can omit STEMMER keyword from the WHERE clause.

```sql
=> SELECT * FROM text_index WHERE token = v_txtindex.StemmerCaseInsensitive('<WARNING>');
```

<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;warning&gt;</td>
<td>2</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;warning&gt;</td>
<td>5</td>
</tr>
</tbody>
</table>
Next, write a query to display the full contents of the source table that match the keyword you searched for in the text index.

```sql
=> SELECT * FROM t_log WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseInsensitive('WARNING'));
```

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2014-06-04</td>
<td>11:00:49.568 unknown:0x7f9207607700 [Catalog] &lt;WARNING&gt; validateDependencies: Object 45935968</td>
</tr>
<tr>
<td>5</td>
<td>2014-06-04</td>
<td>11:00:49.568 unknown:0x7f9207607700 [Catalog] &lt;WARNING&gt; validateDependencies: Object 45930</td>
</tr>
<tr>
<td>2</td>
<td>2013-06-04</td>
<td>11:00:49.568 unknown:0x7f9207607700 [Catalog] &lt;WARNING&gt; validateDependencies: Object 4593</td>
</tr>
<tr>
<td>3</td>
<td>2013-06-04</td>
<td>11:00:49.568 unknown:0x7f9207607700 [Catalog] &lt;WARNING&gt; validateDependencies: Object 45966</td>
</tr>
</tbody>
</table>

Use the doc_id to find the exact location of the keyword in the source table. The doc_id matches the unique identifier from the source table. This matching allows you to quickly find the instance of the keyword in your table.

### Performing a Case-Sensitive and Case-Insensitive Text Search Query

Your text index is optimized to match all instances of words depending upon your stemmer. By default, the case insensitive stemmer is applied to all text indices that do not specify a stemmer. Therefore, if the queries you plan to write against your text index are case sensitive, then Vertica recommends you use a case sensitive stemmer to build your text index.

The following examples show queries that match case-sensitive and case-insensitive words that you can use when performing a text search.

This query finds case-insensitive records in a case insensitive text index:

```sql
=> SELECT * FROM t_log WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseInsensitive('warning'));
```

This query finds case-sensitive records in a case sensitive text index:

```sql
=> SELECT * FROM t_log_case_sensitive WHERE id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive('Warning'));
```
Including and Excluding Keywords in a Text Search Query

Your text index also allows you to perform more detailed queries to find multiple keywords or omit results with other keywords. The following example shows a more detailed query that you can use when performing a text search.

In this example, t_log is the source table, and text_index is the text index. The query finds records that either contain:

- Both the words '<WARNING>' and 'validate'
- Only the word '[Log]' and does not contain 'validateDependencies'

```
SELECT * FROM t_log where (
    id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive('<WARNING>'))
    AND ( id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive ('validate'))
    OR id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive ('[Log]'))
    AND NOT (id IN (SELECT doc_id FROM text_index WHERE token = v_txtindex.StemmerCaseSensitive ('validateDependencies')))));
```

This query returns the following results:

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>2014-05-04</td>
<td>11:00:49.568 unknown:0x7f9207607702 [Log] &lt;WARNING&gt; validate: Object 4503 via fld num_all_roles</td>
</tr>
<tr>
<td>13</td>
<td>2014-05-04</td>
<td>11:00:49.568 unknown:0x7f9207607706 [Log] &lt;WARNING&gt; validate: Object 45035 refers to root_i3</td>
</tr>
<tr>
<td>14</td>
<td>2014-05-04</td>
<td>11:00:49.568 unknown:0x7f9207607708 [Log] &lt;WARNING&gt; validate: Object 4503 refers to int_2</td>
</tr>
<tr>
<td>17</td>
<td>2014-05-04</td>
<td>11:00:49.568 unknown:0x7f9207607700 [Txn] &lt;WARNING&gt; Begin validate Txn: fff0ed17 catalog editor (4 rows)</td>
</tr>
</tbody>
</table>

Dropping a Text Index

Dropping a text index removes the specified text index from the database.

You can drop a text index when:
• It is no longer queried frequently.

• An administrative task needs to be performed on the source table and requires the text index to be dropped.

Dropping the text index does not drop the source table associated with the text index. However, if you drop the source table associated with a text index, then that text index is also dropped. Vertica considers the text index a dependent object.

The following example illustrates how to drop a text index named text_index:

```sql
=> DROP TEXT INDEX text_index;
DROP INDEX
```

## Stemmers and Tokenizers

Vertica provides default stemmers and tokenizers. You can also create your own custom stemmers and tokenizers. The following topics explain the default stemmers and tokenizers, and the requirements for creating custom stemmers and tokenizers in Vertica.

• **Vertica Stemmers**

• **Vertica Tokenizers**

• **Configuring a Tokenizer**

• **Requirements for Custom Stemmers and Tokenizers**

### Vertica Stemmers

Vertica *stemmers* use the Porter stemming algorithm to find words derived from the same base/root word. For example, if you perform a search on a text index for the keyword *database*, you might also want to get results containing the word *databases*.

To achieve this type of matching, Vertica stores words in their stemmed form when using any of the v_txtindex stemmers.

The Vertica Analytics Platform provides the following stemmers:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_txtindex.Stemmer(long varchar)</td>
<td>Not sensitive to case; outputs lowercase words.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Stems strings from a Vertica table. Alias of StemmerCaseInsensitive.</td>
<td></td>
</tr>
<tr>
<td>v_txtindex.StemmerCaseSensitive(long varchar)</td>
<td>Sensitive to case. Stems strings from a Vertica table.</td>
</tr>
<tr>
<td>v_txtindex.StemmerCaseInsensitive (long varchar)</td>
<td>Default stemmer used if no stemmer is specified when creating a text index. Not sensitive to case; outputs lowercase words. Stems strings from a Vertica table.</td>
</tr>
<tr>
<td>v_txtindex.caseInsensitiveNoStemming (long varchar)</td>
<td>Not sensitive to case; outputs lowercase words. Does not use the Porter Stemming algorithm.</td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how to use a stemmer when creating a text index.

Create a text index using the StemmerCaseInsensitive stemmer:

```sql
=> CREATE TEXT INDEX idx_100 ON top_100 (id, feedback) STEMMER v_txtindex.StemmerCaseInsensitive(long varchar) (long varchar);
```

Create a text index using the StemmerCaseSensitive stemmer:

```sql
=> CREATE TEXT INDEX idx_unstruc ON unstruc_data (__identity__, __raw__) STEMMER v_txtindex.StemmerCaseSensitive(long varchar) public.FlexTokenizer(long varbinary);
```

Create a text index without using a stemmer:

```sql
=> CREATE TEXT INDEX idx_logs FROM sys_logs ON (id, message) STEMMER NONE TOKENIZER v_txtindex.StringTokenizer(long varchar);
```

**Vertica Tokenizers**

The Vertica Analytics Platform provides the following pre-configured tokenizers:
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public.FlexTokenizer(long varbinary)</td>
<td>Splits the values in your Flex Table by white space.</td>
</tr>
<tr>
<td>v_txtindex.StringTokenizer(long varchar)</td>
<td>Splits the string into words by splitting on white space.</td>
</tr>
<tr>
<td>v_txtindex.AdvancedLogTokenizer</td>
<td>Uses the default parameters for all tokenizer parameters. For more information, see Advanced Log Tokenizer.</td>
</tr>
<tr>
<td>v_txtindex.BasicLogTokenizer</td>
<td>Uses the default values for all tokenizer parameters except minorseparator, which is set to an empty list. For more information, see Basic Log Tokenizer.</td>
</tr>
<tr>
<td>v_txtindex.WhitespaceLogTokenizer</td>
<td>Uses default values for tokenizer parameters, except for majorseparators, which uses E' \t\n\f\r'; and minorseparator, which uses an empty list. For more information, see Whitespace Log Tokenizer.</td>
</tr>
</tbody>
</table>

Vertica also provides the following tokenizer, which is not pre-configured:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_txtindex.ICUTokenizer</td>
<td>Supports multiple languages. Tokenizes based on the conventions of the language you set in the locale parameter. For more information, see ICU Tokenizer.</td>
</tr>
</tbody>
</table>

### Examples

The following examples show how you can use a pre-configured tokenizer when creating a text index.

Use the StringTokenizer to create an index from the top_100:

```sql
=> CREATE TEXT INDEX idx_100 FROM top_100 on (id, feedback)
    TOKENIZER v_txtindex.StringTokenizer(long varchar)
    STEMMER v_txtindex.StemmerCaseInsensitive(long varchar);
```

Use the FlexTokenizer to create an index from unstructured data:

```sql
=> CREATE TEXT INDEX idx_unstruc FROM unstruc_data on (__identity__, __raw__)
    TOKENIZER public.FlexTokenizer(long varbinary)
    STEMMER v_txtindex.StemmerCaseSensitive(long varchar);
```
Advanced Log Tokenizer

Returns tokens that can include minor separators. You can use this tokenizer in situations when your tokens are separated by whitespace or various punctuation. The advanced log tokenizer offers more granularity than the basic log tokenizer in defining separators through the addition of minor separators. This approach is frequently appropriate for analyzing log files.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using `REBALANCE_CLUSTER` before using a Vertica tokenizer.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopwordscaseinsensitive</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>minorseparators</td>
<td>E'/:=@-$#_</td>
</tr>
<tr>
<td>majorseparators</td>
<td>E' []&lt;&gt;(){</td>
</tr>
<tr>
<td>minLength</td>
<td>'2'</td>
</tr>
<tr>
<td>maxLength</td>
<td>'128'</td>
</tr>
<tr>
<td>used</td>
<td>'True'</td>
</tr>
</tbody>
</table>

Examples

The following example shows how you can create a text index, from the table foo, using the Advanced Log Tokenizer without a stemmer.

```sql
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL, text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
>> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 9986454 for outside:101.123.123.111/443 (101.123.123.111/443)
>> ".
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id
SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_AdvancedLogTokenizer ON foo (id, text)
TOKENIZER v_txtindex.AdvancedLogTokenizer(LONG VARCHAR) STEMMER NONE;
=> SELECT * FROM indexfoo_AdvancedLogTokenizer;
<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ASA-6-382013:</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Basic Log Tokenizer

Returns tokens that exclude specified minor separators. You can use this tokenizer in situations when your tokens are separated by whitespace or various punctuation. This approach is frequently appropriate for analyzing log files.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using REBALANCE_CLUSTER before using a Vertica tokenizer.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopwordscaseinsensitive</td>
<td>''</td>
</tr>
<tr>
<td>minorseparators</td>
<td>''</td>
</tr>
<tr>
<td>majorseparators</td>
<td>E' []&lt;&gt;{}</td>
</tr>
<tr>
<td>minLength</td>
<td>'2'</td>
</tr>
<tr>
<td>maxLength</td>
<td>'128'</td>
</tr>
<tr>
<td>used</td>
<td>'True'</td>
</tr>
</tbody>
</table>
Examples

The following example shows how you can create a text index, from the table foo, using the Basic Log Tokenizer without a stemmer.

```
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL,text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
>> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 9986454 for outside:101.123.123.111/443 (101.123.123.111/443)
>> 
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id  
SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_BasicLogTokenizer ON foo (id, text)  
TOKENIZER v_txtindex.BasicLogTokenizer(LONG VARCHAR) STEammer NONE;
=> SELECT * FROM indexfoo_BasicLogTokenizer;
```

<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ASA-6-302013:</td>
<td>1</td>
</tr>
<tr>
<td>00:00:05.700433</td>
<td>1</td>
</tr>
<tr>
<td>101.123.123.111/443</td>
<td>1</td>
</tr>
<tr>
<td>2014-05-10</td>
<td>1</td>
</tr>
<tr>
<td>9986454</td>
<td>1</td>
</tr>
<tr>
<td>Built</td>
<td>1</td>
</tr>
<tr>
<td>TCP</td>
<td>1</td>
</tr>
<tr>
<td>connection</td>
<td>1</td>
</tr>
<tr>
<td>for</td>
<td>1</td>
</tr>
<tr>
<td>outbound</td>
<td>1</td>
</tr>
<tr>
<td>outside:101.123.123.111/443</td>
<td>1</td>
</tr>
</tbody>
</table>
(11 rows)

Whitespace Log Tokenizer

Returns only tokens surrounded by whitespace. You can use this tokenizer in situations where you want to the tokens in your source document to be separated by whitespace characters only. This approach lets you retain the ability to set stop words and token length limits.

Important: If you create a database with no tables and the k-safety has increased, you must rebalance your data using REBALANCE_CLUSTER before using a Vertica tokenizer.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopwordscaseinsensitive</td>
<td>' '</td>
</tr>
<tr>
<td>minorseparators</td>
<td>' '</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>major separators</td>
<td>E' \t \n \f \r'</td>
</tr>
<tr>
<td>minLength</td>
<td>'2'</td>
</tr>
<tr>
<td>maxLength</td>
<td>'128'</td>
</tr>
<tr>
<td>used</td>
<td>'True'</td>
</tr>
</tbody>
</table>

**Examples**

The following example shows how you can create a text index, from the table `foo`, using the Whitespace Log Tokenizer without a stemmer.

```sql
=> CREATE TABLE foo (id INT PRIMARY KEY NOT NULL,text VARCHAR(250));
=> COPY foo FROM STDIN;
End with a backslash and a period on a line by itself.
=> 1|2014-05-10 00:00:05.700433 %ASA-6-302013: Built outbound TCP connection 998 6454 for outside:101.123.123.111/443 (101.123.123.111/443)
=> \.
=> CREATE PROJECTION foo_projection AS SELECT * FROM foo ORDER BY id
SEGMENTED BY HASH(id) ALL NODES KSAFE;
=> CREATE TEXT INDEX indexfoo_WhitespaceLogTokenizer ON foo (id, text)
  TOKENIZER v_txtindex.WhitespaceLogTokenizer(LONG VARCHAR) STEMER NONE;
=> SELECT * FROM indexfoo_WhitespaceLogTokenizer;
```

<table>
<thead>
<tr>
<th>token</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ASA-6-302013:</td>
<td>1</td>
</tr>
<tr>
<td>(101.123.123.111/443)</td>
<td>1</td>
</tr>
<tr>
<td>00:00:05.700433</td>
<td>1</td>
</tr>
<tr>
<td>2014-05-10</td>
<td>1</td>
</tr>
<tr>
<td>6454</td>
<td>1</td>
</tr>
<tr>
<td>998</td>
<td>1</td>
</tr>
<tr>
<td>Built</td>
<td>1</td>
</tr>
<tr>
<td>TCP</td>
<td>1</td>
</tr>
<tr>
<td>connection</td>
<td>1</td>
</tr>
<tr>
<td>for</td>
<td>1</td>
</tr>
<tr>
<td>outbound</td>
<td>1</td>
</tr>
<tr>
<td>outside:101.123.123.111/443</td>
<td>1</td>
</tr>
</tbody>
</table>

**ICU Tokenizer**

Supports multiple languages. You can use this tokenizer to identify word boundaries in languages other than English, including Asian languages that are not separated by whitespace.

The ICU Tokenizer is not pre-configured. You configure the tokenizer by first creating a User-Defined Transform Function (UDTF). Then set the parameter, locale, to identify the language to tokenizer.
**Important:** If you create a database with no tables and the k-safety has increased, you must rebalance your data using `REBALANCE_CLUSTER` before using a Vertica tokenizer.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>Uses the POSIX naming convention: <code>language[_COUNTRY]</code>&lt;br&gt;Identify the language using its ISO-639 code, and the country using its ISO-3166 code. For example, the parameter value for simplified Chinese is <code>zh_CN</code>, and the value for Spanish is <code>es_ES</code>&lt;br&gt;The default value is English if you do not specify a locale.</td>
</tr>
</tbody>
</table>

### Example

The following example steps show how you can configure the ICU Tokenizer for simplified Chinese, then create a text index from the table foo, which contains Chinese characters.

For more on how to configure tokenizers, see [Configuring a Tokenizer](#).

1. **Create the tokenizer using a UDTF.** The example tokenizer is named `ICUChineseTokenizer`.

   ```sql
   VMart=> CREATE OR REPLACE TRANSFORM FUNCTION v_txtindex.ICUChineseTokenizer AS LANGUAGE 'C++'
   NAME 'ICUTokenizerFactory' LIBRARY v_txtindex.logSearchLib NOT FENCED;
   CREATE TRANSFORM FUNCTION
   ```

2. **Get the procedure ID of the tokenizer.**

   ```sql
   VMart=> SELECT proc_oid from vs_procedures where procedure_name = 'ICUChineseTokenizer';
   proc_oid
   ------------------------
   45035996280452894
   (1 row)
   ```

3. **Set the parameter, locale, to simplified Chinese.** Identify the tokenizer using its procedure ID.

   ```sql
   VMart=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('locale','zh_CN' using parameters proc_oid='45035996280452894');
   SET_TOKENIZER_PARAMETER
   ------------------------
   t
   ```
4. Lock the tokenizer.

```sql
VMart=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('used','true' using parameters proc_oid='45835996273762696');
SET_TOKENIZER_PARAMETER
--------------
t
(1 row)
```

5. Create an example table, foo, containing simplified Chinese text to index.

```sql
VMart=> CREATE TABLE foo(doc_id integer primary key not null,text varchar(250));
CREATE TABLE
VMart=> INSERT INTO foo values(1, u\4E2D\534E\4EBA\6C11\5171\548C\56FD');
OUTPUT
-------
1
```

6. Create an index, index_example, on the table foo. The example creates the index without a stemmer; Vertica stemmers work only on English text. Using a stemmer for English on non-English text can cause incorrect tokenization.

```sql
VMart=> CREATE TEXT INDEX index_example ON foo (doc_id, text) TOKENIZER v_txtindex.ICUChineseTokenizer(long varchar) stemmer none;
CREATE INDEX
```

7. View the new index.

```sql
VMart=> SELECT * FROM index_example ORDER BY token,doc_id;
token | doc_id
-------+--------
中华  | 1
人民  | 1
共和国| 1
(3 rows)
```

### Configuring a Tokenizer

You configure a tokenizer by creating a User-Defined Transform Function (UDTF) using one of the two base UDTFs in the v_txtindex.AdvTxtSearchLib library. The library contains two base tokenizers: one for Log Words and one for Ngrams. You can configure each base function with or without positional relevance.

You can choose among several different tokenizer base configurations:
Create a logWord tokenizer without positional relevance:

```sql
=> CREATE TRANSFORM FUNCTION v_txtindex.fooTokenizer AS LANGUAGE 'C++' NAME 'logWordITokenizerFactory' LIBRARY v_txtindex.logSearchLib NOT FENCED;
```

### Retrieve a Tokenizer's proc_oid

After you create the tokenizer, Vertica writes the name and proc_oid to the system table `vs_procedures`. You must retrieve the tokenizer's proc_oid to perform additional configuration.

Enter the following query, substituting your own tokenizer name:

```sql
=> SELECT proc_oid FROM vs_procedures WHERE procedure_name = 'fooTokenizer';
```

### Set Tokenizer Parameters

Use the tokenizer's proc_oid to configure the tokenizer. See [Configuring a Tokenizer](#) for more information about getting the proc_oid of your tokenizer. The following examples show how you can configure each of the tokenizer parameters:

#### Configure stop words:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('stopwordscaseinsensitive', 'for, the' USING PARAMETERS proc_oid='45035996274128376');
```

#### Configure major separators:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('majorseparators', E'{}()'&[]' USING PARAMETERS proc_oid='45035996274128376');
```

#### Configure minor separators:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('minorseparators', '-,$' USING PARAMETERS proc_oid='45035996274128376');
```

#### Configure minimum length:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('minlength', '1' USING PARAMETERS proc_oid='45035996274128376');
```
Configure maximum length:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('maxlength', '140' USING PARAMETERS proc_oid='45035996274128376');
```

Configure ngramssize:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('ngramssize', '2' USING PARAMETERS proc_oid='45035996274128376');
```

Lock Tokenizer Parameters

When you finish configuring the tokenizer, set the parameter, used, to True. After changing this setting, you are no longer able to alter the parameters of the tokenizer. At this point, the tokenizer is ready for you to use to create a text index.

Configure the used parameter:

```sql
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('used', 'True' USING PARAMETERS proc_oid='45035996274128376');
```

See Also

`SET_TOKENIZER_PARAMETER`

View Tokenizer Parameters

After creating a custom tokenizer, you can view the tokenizer's parameter settings in either of two ways:

- Use the `GET_TOKENIZER_PARAMETER` — View individual tokenizer parameter settings.
- Use the `READ_CONFIG_FILE` — View all tokenizer parameter settings.

View Individual Tokenizer Parameter Settings

If you need to see an individual parameter setting for a tokenizer, you can use `GET_TOKENIZER_PARAMETER` to see specific tokenizer parameter settings:

```sql
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('majorseparators' USING PARAMETERS proc_oid='45035996274126984');
```
For more information, see GET_TOKENIZER_PARAMETER.

View All Tokenizer Parameter Settings

If you need to see all of the parameters for a tokenizer, you can use READ_CONFIG_FILE to see all of the parameter settings for your tokenizer:

```sql
=> SELECT v_txtindex.READ_CONFIG( USING PARAMETERS proc_oid='45035996274126984') OVER();

<table>
<thead>
<tr>
<th>config_key</th>
<th>config_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>majorseparators</td>
<td>{},[]</td>
</tr>
<tr>
<td>maxlength</td>
<td>140</td>
</tr>
<tr>
<td>minlength</td>
<td>1</td>
</tr>
<tr>
<td>minorseparators</td>
<td>-,$</td>
</tr>
<tr>
<td>stopwordscaseinsensitive</td>
<td>for,the</td>
</tr>
<tr>
<td>type</td>
<td>1</td>
</tr>
<tr>
<td>used</td>
<td>true</td>
</tr>
</tbody>
</table>
```

(7 rows)

If the parameter, used, is set to False, then you can only view the parameters that have been applied to the tokenizer.

**Note:** Vertica automatically supplies the value for Type, unless you are using an ngram tokenizer, which allows you to set it.

For more information, see READ_CONFIG_FILE.

Delete Tokenizer Config File

Use the DELETE_TOKENIZER_CONFIG_FILE function to delete a tokenizer configuration file. This function does not delete the User- Defined Transform Function (UDTF). It only deletes the configuration file associated with the UDTF.

Delete the tokenizer configuration file when the parameter, used, is set to False:

```sql
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274127086');
```

Delete the tokenizer configuration file with the parameter, confirm, set to True. This setting forces the configuration file deletion, even if the parameter, used, is also set to True:

```sql
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274126984', confirm='true');
```

For more information, see DELETE_TOKENIZER_CONFIG_FILE.
Requirements for Custom Stemmers and Tokenizers

Sometimes, you may want specific tokenization or stemming behavior that differs from what Vertica provides. In such cases, you can implement your own custom User Defined Extensions (UDx) to replace the stemmer or tokenizer. For more information about building custom UDxs see Developing User-Defined Extensions (UDxs).

Before implementing a custom stemmer or tokenizer in Vertica verify that the UDx extension meets these requirements.

**Note:** Custom tokenizers can return multi-column text indices.

Vertica Stemmer Requirements

Comply with these requirements when you create custom stemmers:

- Must be a User Defined Scalar Function (UDSF) or a SQL Function
- Can be written in C++, Java, or R
- Volatility set to stable or immutable

**Supported Data Input Types:**

- Varchar
- Long varchar

**Supported Data Output Types:**

- Varchar
- Long varchar

Vertica Tokenizer Requirements

To create custom tokenizers, follow these requirements:

- Must be a User Defined Transform Function (UDTF)
- Can be written in C++, Java, or R
• Input type must match the type of the input text

Supported Data Input Types:

• Char
• Varchar
• Long varchar
• Varbinary
• Long varbinary

Supported Data Output Types:

• Varchar
• Long varchar
Bulk-Loading Data

This section describes different methods to bulk-load data into a Vertica database using the COPY statement. In its simplest form, COPY copies data from a source to a file, as follows:

```
COPY target-table FROM data-source
```

Source data can be a data stream or a file path. You can specify many details about a data load, including:

- Data format and compression
- Which built-in parser to use, or which user-defined source, filters, or parser to use
- How to distribute the data load among database nodes
- How to transform data during loading
- What to do with data that could not be loaded

Checking Data Format Before or After Loading

Vertica supports loading data files in the Unicode UTF-8 format. You can load ASCII data, which is UTF-8 compatible. Character sets like ISO 8859-1 (Latin1) are incompatible with UTF-8 and are not supported.

Before loading data from text files, you can use several Linux tools to ensure that your data is in UTF-8 format. The `file` command reports the encoding of any text files. For example:

```
$ file Date_Dimension.tbl
Date_Dimension.tbl: ASCII text
```

The `file` command could indicate ASCII TEXT even though the file contains multibyte characters.

To check for multibyte characters in an ASCII file, use the `wc` command. For example:

```
$ wc Date_Dimension.tbl
1828 5484 221822 Date_Dimension.tbl
```

If the `wc` command returns an error such as Invalid or incomplete multibyte or wide character, the data file is using an incompatible character set.
This example describes files that are not UTF-8 data files. Two text files have filenames starting with the string data. To check their format, use the `file` command as follows:

```
$ file data*
  data1.txt: Little-endian UTF-16 Unicode text
  data2.txt: ISO-8859 text
```

The results indicate that neither of the files is in UTF-8 format.

## Converting Files Before Loading Data

To convert files before loading them into Vertica, use the `iconv` UNIX command. For example, to convert the `data2.txt` file from the previous example, use the `iconv` command as follows:

```
iconv -f ISO88599 -t utf-8 data2.txt > data2-utf8.txt
```

See the man pages for `file` and `iconv` for more information.

## Checking UTF-8 Compliance After Loading Data

After loading data, use the `ISUTF8` function to verify that all of the string-based data in the table is in UTF-8 format. For example, if you loaded data into a table named `nametable` that has a `VARCHAR` column named `name`, you can use this statement to verify that all of the strings are UTF-8 encoded:

```
=> SELECT name FROM nametable WHERE ISUTF8(name) = FALSE;
```

If all of the strings are in UTF-8 format, the query should not return any rows.

## Performing the Initial Database Load

To perform the initial database load, use `COPY` with its `DIRECT` parameter from `vsql`.

Tip: Vertica supports multiple schema types. If you have a star schema, load the smaller tables before you load the largest tables.

Only a superuser can use the `COPY` statement to bulk load data. Two exceptions to the superuser requirement are to:
1. Run COPY to load from a stream on the host (such as STDIN) rather than a file (see Streaming Data Via JDBC).

2. Use the COPY statement with the FROM LOCAL option.

A non-superuser can also perform a batch load with a JDBC prepared statement, which invokes COPY to load data as a background task.

Extracting Data From an Existing Database

If possible, export the data in text form to a local file or attached disk. When working with large amounts of load data (> 500GB), Vertica recommends that you test the load process using smaller load files as described in Configuration Procedure to avoid compatibility or file formatting issues.

ETL products typically use ODBC or JDBC to extract data, which gives them program-level access to modify load file column values, as needed.

Database systems typically provide a variety of export methods.

Tip: To export data from an Oracle database, run a SELECT query in Oracle’s SQL*Plus command line query tool using a specified column delimiter, suppressed headers, and so forth. Redirect the output to a local file.

Smaller tables generally fit into a single load file. Split any large tables into 250-500GB load files. For example, a 10 TB fact table will require 20-40 load files to maintain performance.

Checking for Delimiter Characters in Load Data

The default delimiter for the COPY statement is a vertical bar (|). Before loading your data, make sure that no CHAR(N) or VARCHAR(N) data values include the delimiter character.

To test for the existence of a specific character in a column, use a query such as this:

```
SELECT COUNT(*) FROM T WHERE X LIKE '%|%'
```

If only a few rows contain |, you can eliminate them from the load file using a WHERE clause and load them separately using a different delimiter.

Tip: For loading data from an Oracle database, use a WHERE clause to avoid problem rows in the main load file, and the negated WHERE clause with REGEX_REPLACE for problem rows.
Moving Data From an Existing Database to Vertica Nodes

To move data from an existing database to Vertica, consider using:

- USB 2.0 (or possibly SATA) disks
- A fast local network connection

Deliver chunks of data to the different Vertica nodes by connecting the transport disk or by writing files from network copy.

Loading From a Local Hard Disk

USB 2.0 disks can deliver data at about 30 MB per second, or 108 GB per hour. USB 2.0 disks are easy to use for transporting data from Linux to Linux. Set up an ext4 filesystem on the disk and write large files there. Linux 2.6 has USB plug-and-play support, so a USB 2.0 disk is instantly usable on various Linux systems.

For other UNIX variants, if there is no common filesystem format available, use the disk without a filesystem to copy a single large file. For example:

```
$ cp bigfile /dev/sdc1
```

Even without a filesystem on the disk, plug-and-play support still works on Linux to provide a device node for the disk. To find out the assigned device, plug in the disk and enter:

```
$ dmesg | tail -40
```

SATA disks are usually internal, but can be external, or unmounted safely if they are internal.

Loading Over the Network

A 1Gbps (gigabits per second) network can deliver about 50 MB/s, or 180GB/hr. Vertica can load about 30-50GB/hour/node for a 1-Ksafe projection design. Therefore, you should use a dedicated 1Gbps LAN. Using a LAN with a performance that is < 1Gbps will be proportionally slower. Vertica recommends not loading data across an external network, because the delays over distance slow down the TCP protocol to a small fraction of its available bandwidth, even without competing traffic.
Note: The actual load rates you obtain can be higher or lower depending on the properties of the data, number of columns, number of projections, and hardware and network speeds. Load speeds can be further improved by using multiple parallel streams.

Loading From Windows

Use NTFS for loading files directly from Windows to Linux. Although Red Hat Linux as originally installed can read Windows FAT32 file systems, this is not recommended.

Using Load Scripts

You can write and run a load script for the COPY statement using a simple text-delimited file format. For information about other load formats see Specifying a COPY Parser. Vertica recommends that you load the smaller tables before the largest tables. To check data formats before loading, see Checking Data Format Before or After Loading.

Using Absolute Paths in a Load Script

Unless you are using the COPY FROM LOCAL statement, using COPY on a remote client requires an absolute path for a data file. You cannot use relative paths on a remote client. For a load script, you can use vsql variables to specify the locations of data files relative to your Linux working directory.

To use vsql variables to specify data file locations:

1. Create a vsql variable containing your Linux current directory.

\set t_pwd \pwd

2. Create another vsql variable that uses a path relative to the Linux current directory variable for a specific data file.

\set input_file "':t_pwd'/Date_Dimension.tbl"

3. Use the second variable in the COPY statement:
4. Repeat steps 2 and 3 to load all data files.

Note: COPY FROM LOCAL does not require an absolute path for data files. You can use paths that are relative to the client's directory system.

Running a Load Script

You can run a load script on any host, as long as the data files are on that host.

1. Change your Linux working directory to the location of the data files.

\[
\text{$ cd /opt/vertica/doc/retail_example_database}
\]

2. Run the Administration Tools.

\[
\text{$ /opt/vertica/bin/admintools}
\]

3. Connect to the database.

4. Run the load script.

Loading Data Interactively

Vertica recommends using the COPY statement in script files, as described in Using Load Scripts. You can also load data interactively by piping a text file to vsql and executing a COPY (or COPY FROM LOCAL) statement with the standard input stream as the input file.

For example:

\[
\text{$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM STDIN DELIMITER '|' DIRECT";}
\]
\[
\text{$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM LOCAL STDIN DELIMITER '|' DIRECT";}
\]
Using COPY and COPY LOCAL

The COPY statement bulk loads data into a Vertica database. You can initiate loading one or more files or pipes on a cluster host. You can also load directly from a client system, using the COPY statement with its FROM LOCAL option.

COPY lets you load parsed or computed data.Parsed data is from a table or schema using one or more columns, and computed data is calculated with a column expression on one or more column values.

COPY invokes different parsers depending on the data format you specify:

- Delimited text (the default parser format, but never specified)
- Native binary (NATIVE) (not supported with COPY LOCAL)
- Native varchar (NATIVE VARCHAR) (not supported with COPY LOCAL)
- Fixed-width data (FIXEDWIDTH)
- ORC (Optimized Row Columnar) and Parquet Hadoop files (not supported with COPY LOCAL)

To use one of the several flex table parsers, use the parser parameter, followed by the parser of choice.

COPY has many options, which you can combine to make importing data flexible. For detailed syntax of the various options see the SQL Reference Manual. The Administrator's Guide describes some common operations as follows:

<table>
<thead>
<tr>
<th>For this option...</th>
<th>See this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncompressed data, or data in GZIP, BZIP, or LZO compressed forms.</td>
<td>Specifying COPY FROM Options</td>
</tr>
<tr>
<td>Insert data into the WOS (memory) or directly into the ROS (disk).</td>
<td>Choosing a Load Method</td>
</tr>
<tr>
<td>Set parameters such as data delimiters and quote characters for the entire load operation or, for specific columns.</td>
<td>Loading Delimited Data</td>
</tr>
<tr>
<td>Transform data before inserting it into the database.</td>
<td>Transforming Data During Loads</td>
</tr>
</tbody>
</table>
Copying Data from a Vertica Client

Use `COPY LOCAL` to load files on a client system to the Vertica database. For example, to copy a GZIP file from your local client, use a command such as this:

```plaintext
=> COPY store.store_dimension FROM LOCAL '/usr/files/my_data/input_file' GZIP;
```

You can use a comma-separated list to load multiple files of the same compression type. `COPY LOCAL` then concatenates the files into a single file, so you cannot combine files with different compression types in the list. When listing multiple files, be sure to specify the type of every input file, such as BZIP, as shown:

```plaintext
=>COPY simple_table FROM LOCAL 'input_file.bz' BZIP, 'input_file.bz' BZIP;
```

You can load data from a local client from STDIN, as follows:

```plaintext
=> COPY simple_table FROM LOCAL STDIN;
```

Loading Data from an IDOL CFS Client

The IDOL Connector Framework Server (CFS) VerticalIndexer feature lets CFS clients connect to your Vertica database using ODBC. After it is connected, CFS uses `COPY...FROM LOCAL` statements to load IDOL document metadata into an existing flex table. For more information, see the Using Flex Tables for IDOL Data section in Using Flex Tables.

Transforming Data During Loads

To promote a consistent database and reduce the need for scripts to transform data at the source, you can transform data with an expression as part of loading. Transforming data while loading is useful for computing values to insert into a target database column from other columns in the source database.

To transform data during load, use the following syntax to specify the target column for which you want to compute values, as an expression:

```plaintext
COPY [ [database-name.]schema-name.]table [[[Column as Expression] / column[FORMAT 'format']
    [, ....]]]
FROM ...
Understanding Transformation Requirements

When transforming data during loads, the COPY statement must contain at least one parsed column. The parsed column can be a FILLER column.

Specify only RAW data in the parsed column source data. If you specify nulls in that RAW data, the columns are evaluated with the same rules as for SQL statement expressions.

You can intersperse parsed and computed columns in a COPY statement.

Loading FLOAT Values

Vertica parses floating-point values internally. COPY does not require you to cast floats explicitly, unless you need to transform the values for another reason.

Using Expressions in COPY Statements

The expression in a COPY statement can be as simple as a single column, or more complex, such as a case statement for multiple columns. An expression can specify multiple columns, and multiple expressions can refer to the same parsed column. You can use expressions for columns of all supported data types.

COPY expressions can use many Vertica-supported SQL functions, operators, constants, NULLs, and comments, including these functions:

- Date/time
- Formatting Functions
- String
- Null-handling
- System information

Requirements and restrictions

- COPY expressions cannot use SQL meta-functions (Vertica-specific), analytic functions, aggregate functions, or computed columns.
For computed columns, you must list all parsed columns in the COPY statement expression. Do not specify \texttt{FORMAT} or \texttt{RAW} in the source data for a computed column.

Expressions used in a COPY statement can contain only constants. The return data type of the expression must be coercible to that of the target column. Parsed column parameters are also coerced to match the expression.

## Handling Expression Errors

Errors in expressions within your COPY statement are SQL errors. As such, they are handled differently from parse errors. When a parse error occurs, COPY rejects the row and adds it to the rejected data file or table. COPY also adds the reason for a rejected row to the exceptions file or the rejected data table. For example, COPY parsing does not implicitly cast data types. If a type mismatch occurs between the data being loaded and a column type (such as attempting to load a text value into a \texttt{FLOAT} column), COPY rejects the row, and continues processing.

If an error occurs in an expression in your COPY statement, then by default the entire load fails. For example, if your COPY statement has a transform function expression, and a syntax error exists in that expression, the entire load is rolled back. All SQL errors, including COPY rollback from an expression, are stored in the Vertica-specific log file. However, unlike parse rejections and exception messages, SQL expression errors are brief, and may require further research.

You can have COPY treat errors in transformation expressions like parse errors. Rejected rows are added to the same file or table, and exceptions are added to the same exceptions file or table. To enable this behavior, set the \texttt{CopyFaultTolerantExpressions} configuration parameter to 1. (See \texttt{General Parameters}.)

Loading data with expression rejections is potentially slower than loading with the same number of parse rejections. Enable expression rejections if your data has a few bad rows; doing so allows the rest of the data to be loaded. If you are concerned about the time it takes to complete a load with many bad rows, use the \texttt{REJECTMAX} parameter to set a limit. If \texttt{COPY} finds more than \texttt{REJECTMAX} bad rows, it aborts and rolls back the load.

See \texttt{Capturing Load Rejections and Exceptions} for more information about managing rejected data.

## Transformation Example

Following is a small transformation example.
1. Create a table `t`.

```sql
CREATE TABLE t (
    year VARCHAR(10),
    month VARCHAR(10),
    day VARCHAR(10),
    k timestamp
);```

2. Use COPY to copy the table, computing values for the year, month, and day columns in the target database, based on the timestamp columns in the source table.

```sql
COPY t
    (year AS TO_CHAR(k, 'YYYY'),
     month AS TO_CHAR(k, 'Month'),
     day AS TO_CHAR(k, 'DD'),
     k) FORMAT 'YYYY-MM-DD'
FROM STDIN NO COMMIT;
```

```
2009-06-17
1979-06-30
2007-11-26
.
```

3. Load the parsed column, `timestamp`, from the source data to the target database.

```sql
SELECT *
FROM t;
```

```
+-----+-----+-----+---------------+
<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>June</td>
<td>17</td>
<td>2009-06-17 00:00:00</td>
</tr>
<tr>
<td>1979</td>
<td>June</td>
<td>30</td>
<td>1979-06-30 00:00:00</td>
</tr>
<tr>
<td>2007</td>
<td>November</td>
<td>26</td>
<td>2007-11-26 00:00:00</td>
</tr>
</tbody>
</table>
(3 rows)
```

4. Select the table contents to see the results:

```
Deriving Table Columns From Data File Columns

You can use COPY to derive a table column from the data file to load. For more information, see Manipulating Source Data Columns.
```

**Specifying COPY FROM Options**

Each COPY statement requires a FROM option to indicate the location of the file or files being loaded. This syntax excerpt shows the available FROM keywords, and their associated file format options:
Each of the FROM keywords lets you optionally specify the format of the load file as UNCOMPRESSED, BZIP, GZIP, or LZO.

Note: When using COPY in conjunction with a CREATE EXTERNAL TABLE statement, you cannot use the COPY FROM STDIN or LOCAL options.

Some COPY FROM options are not available for all file types. See COPY Parameters.

Loading from STDIN

Using STDIN for the FROM option lets you load uncompressed data, BZIP, or GZIP files.

Loading from a Specific Path

Use the *pathToData* argument to indicate the location of the file to load. You can load data from the following locations:

- The local file system.
- HDFS, using a URL of the form "hdfs://path/to/data". For more information about HDFS URLs, see Reading Directly from HDFS in Integrating with Apache Hadoop.
- An Amazon S3 bucket, for data in text, delimited, Parquet, and ORC formats only. Use a URL of the form 'S3://bucket/path'. See Loading from an S3 Bucket.

You can specify more than one path in the same COPY statement, as in the following example.

```sql
=> COPY myTable FROM 'hdfs:///data/sales/01/*.dat', 'hdfs:///data/sales/02/*.dat', 'hdfs:///data/sales/historical.dat';
```

If *pathToData* resolves to a storage location on a local file system (not HDFS), and the user invoking COPY is not a superuser, these permissions are required:
The storage location must have been created with the USER option (see CREATE LOCATION)

The user must already have been granted READ access to the storage location where the file or files exist, as described in GRANT (Storage Location)

If a user with privileges, who is not a superuser, invokes COPY from that storage location, Vertica prevents symbolic links from allowing unauthorized access.

Loading on Specific Nodes

You can optionally indicate which node or nodes should parse the input by using any of the following:

- A node name
- A set of nodes
- ON ANY NODE

Using the ON ANY NODE clause indicates that the source file to load is available on all of the nodes. If you specify this clause, COPY opens the file and parses it from any node in the cluster. ON ANY NODE is the default for HDFS and S3 paths.

Using the ON nodeset clause indicates that the source file is on all named nodes. If you specify this clause, COPY opens the file and parses it from any node in the set. Be sure that the source file you specify is available and accessible on each applicable cluster node.

Loading Compressed Files (BZIP, GZIP, and LZO)

You can load files compressed with BZIP, GZIP, or LZO. To do so, you must indicate the compression format for each file when loading multiple files. For example, this statement copies a BZIP file into the flex table twitter, using the fjsonparser:

```
=> COPY twitter FROM '/server1/TWITTER/tweets1.json.bz2' BZIP parser fjsonparser() direct;
Rows Loaded
-----------
  172894
(1 row)
```

The following statement loads an LZO file delimited with '|' characters into the flex table twitter:
Vertica supports the following options to the `lzop` command used to compress the file:

- compression level: -1 through -9, --fast, --best
- -F, --no-checksum, --crc32, --adler32

For more information about these options, see [lzop.org](http://lzop.org).

You cannot use BZIP, GZIP, and LZO compression with files in the ORC or Parquet formats.

**Loading with Wildcards (glob)**

You can invoke COPY for a large number of files in a shared directory with a single statement such as:

```sql
=> COPY myTable FROM '/data/manyfiles/*.dat' ON ANY NODE;
```

The glob (*) must indicate a set of files, not directories. The following statement fails if `/data/manyfiles` contains any subdirectories:

```sql
=> COPY myTable FROM '/data/manyfiles/*' ON ANY NODE;
```

Using a wildcard with the ON ANY NODE clause expands the file list on the initiator node. This command then distributes the individual files among all nodes, so that the COPY workload is evenly distributed across the entire cluster.

ON ANY NODE is the default for HDFS paths, as in the following example:

```sql
=> COPY myTable FROM 'hdfs:///data/manyfiles/';
```

You can also distribute a file set across a subset of nodes, which you might do to balance concurrent loads. For example, this command distributes the loading of individual files among the three named nodes:

```sql
=> COPY myTable FROM '/mydirectory/ofmanyfiles/*.dat'
    ON (v_vmart_node0001, v_vmart_node0002, v_vmart_node0003);
```

Distributing file loads across nodes depends on two configuration parameters, `EnableApportionLoad` and `EnableApportionFileLoad`. Both are enabled by default. See [General Parameters](#) for more information about these parameters.
Loading from an S3 Bucket

To access data in S3 you must first do the following tasks:

- Create an IAM role and grant that role permission to access your S3 resources. For more information about IAM roles, see Amazon's Web Services documentation.

- Set the AWSRegion configuration parameter to tell Vertica which AWS region your S3 bucket is in, as in the following example. If the region is not correct, you might experience a delay before the load fails because Vertica retries several times before giving up.

  \[
  \text{=> SELECT SET_CONFIG_PARAMETER('AWSRegion','us-west-1');}
  \]

You might need to set other AWS Parameters to specify a certificate authority.

You can then load data from S3 as in the following example.

  \[
  \text{=> COPY t FROM 's3://AWS_DataLake/sales.parquet' PARQUET;}
  \]

You can specify either a path, as in the previous example, or a glob, if all files in the glob can be loaded together. In the following example, AWS_DataLake contains only ORC files.

  \[
  \text{=> COPY t FROM 's3://AWS_DataLake/*' ORC;}
  \]

You can specify a list of comma-separated S3 buckets as in the following example. All buckets must be in the same region. To load from more than one region, use separate COPY statements and change the value of AWSRegion between calls.

  \[
  \text{=> COPY t FROM 's3://AWS_Data_1/sales.parquet', 's3://AWS_Data_2/sales.parquet' PARQUET;}
  \]

Parquet files can be partitioned, and Vertica can use partitioning information to improve query performance. See Using Partition Columns for more information.

Loading from a Local Client

To bulk-load data from a client, and without requiring database superuser privileges, use the COPY FROM LOCAL option. You can load from either STDIN, or a specific path, but not from a specific node (or ON ANY NODE), since you are loading from the client. All local files are loaded and parsed serially with each COPY statement, so you cannot perform parallel loads with the LOCAL option. See Using Parallel Load Streams.

You can load one or more files in the supported formats: UNCOMPRESSED, BZIP, GZIP, or LZO.
For specific information about saving rejected data and exceptions files when using COPY from LOCAL, see Capturing Load Rejections and Exceptions.

Choosing a Load Method

By default, Vertica loads table data into WOS. After WOS reaches full capacity, Vertica continues to load data to ROS containers. This approach is suitable for small bulk load operations. If desired, you can specify alternative load methods. You can do so at two levels:

- Table metadata, through CREATE TABLE and ALTER TABLE
- DML statements: COPY/COPY FROM VERTICA, INSERT, MERGE, and UPDATE

If a table specifies a load option, Vertica uses it for all DML operations unless the DML statement specifies otherwise.

Load Options and Hints

CREATE TABLE and ALTER TABLE, and copy operations COPY/COPY FROM VERTICA support the following load options:

- AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.
- DIRECT: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.
- TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.

Vertica also supports three load hints: /*+AUTO*/, /*+DIRECT*/, and /*+TRICKLE*/. These hints let you control how individual INSERT, MERGE, and UPDATE operations load table data, overriding the target table’s load setting, if any.

AUTO: Loading Into WOS

If no load option is specified for an operation, Vertica uses the AUTO method to load data into WOS. After WOS reaches full capacity, Vertica continues to load data to ROS containers.
DIRECT: Loading Directly to ROS

The DIRECT option specifies to load data directly into ROS containers, bypassing WOS. DIRECT is best suited for large data loads (100 MB or more).

Note: A large initial bulk load can temporarily affect query performance while Vertica organizes the data.

For example:

COPY a FROM stdin DIRECT;
COPY b FROM LOCAL STDIN DIRECT;

Tip: Avoid using DIRECT to load many smaller data sets. This approach results in many ROS containers that must be combined later.

TRICKLE Loading

Use the TRICKLE load option to load data incrementally after the initial bulk load is complete. Trickle loading loads data into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. Use this option only when the following conditions are true:

- You have a finely tuned load and moveout process at your site.
- You are confident that the WOS can hold the data you are loading.

This option is more efficient than AUTO when you want to load data into partitioned tables.

Overriding COPY Auto Commit

By default, COPY automatically commits itself and other current transactions except when loading temporary tables. You can override this behavior by qualifying the COPY statement with the NO COMMIT option. When you specify NO COMMIT, Vertica does not commit the transaction until you explicitly issue a COMMIT statement.

You can use COPY...NO COMMIT in two ways:
- Execute multiple COPY commands as a single transaction.
- Check data for constraint violations before committing the load.

## Combine Multiple COPY Statements in Same Transaction

When you combine multiple COPY...NO COMMIT statements in the same transaction, Vertica can consolidate the data for all operations into fewer ROS containers, and thereby perform more efficiently.

For example, the following set of COPY...NO COMMIT statements performs several copy statements sequentially, and then commits them all. In this way, all of the copied data is either committed or rolled back as a single transaction.

```sql
COPY... NO COMMIT;
COPY... NO COMMIT;
COPY... NO COMMIT;
COPY X FROM LOCAL NO COMMIT;
COMMIT;
```

**Tip:** Vertica recommends that you COMMIT or ROLLBACK the current transaction before you use COPY...NO COMMIT. Otherwise, if a previous operation such as INSERT is in progress, COPY...NO COMMIT adds that operation to its own transaction. In this case, the previous operation and copy operation are combined as a single transaction, which requires an explicit COMMIT statement.

## Check Constraint Violations

Unless you have enabled enforcement of primary key, unique, or check constraints, Vertica does not check for constraint violations when loading data. You can use COPY...NO COMMIT to troubleshoot loaded data for constraint violations. Before committing the load, test the data with ANALYZE_CONSTRAINTS. If you find any constraint violations, you can roll back the load.

For details on ANALYZE_CONSTRAINTS, see Detecting Constraint Violations with ANALYZE_CONSTRAINTS.

For details on automatic enforcement of primary key, unique, and check constraints, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.
Specifying a COPY Parser

By default, COPY uses the DELIMITER parser to load raw data into the database. Raw input data must be in UTF-8, delimited text format. Data is compressed and encoded for efficient storage.

Note: COPY cannot explicitly specify the DELIMITER parser.

If the raw data to load does not consist primarily of delimited text, specify the parser that is most appropriate, one of the following:

- **NATIVE** (binary)
- **NATIVE VARCHAR**
- **FIXEDWIDTH**
- **ORC** and **PARQUET** (see Reading Hadoop Columnar File Formats)

Using a different parser for your data can improve load performance. If delimited input data includes binary data types, COPY translates the data on input. See Using Load Scripts and Loading Binary (Native) Data for examples. You can also load binary data, but only if it adheres to the COPY format requirements, described in Appendix: Creating Native Binary Format Files.

The same bulk load COPY statement cannot mix raw data types that require different parsers, such as **NATIVE** and **FIXEDWIDTH**. For information about verifying input data formats, see Checking Data Format Before or After Loading.

Flex Table Parsers

You can use flex parsers to load data into standard, columnar tables. Use the flex parser that best matches your needs, as described in the Using Flex Table Parsers section of Using Flex Tables:

- **FAVROPARSER**
- **FCEFPARSER**
- **FCSVPARSER**
- **FDELIMITEDPARSER**
Loading data with the flex parsers makes loading data flexible. For example, you can load JSON data into a columnar table in one load with the `FJSONPARSER`, and delimited data into the same table in another with the default COPY parser. Using Flex Tables describes this use case and presents an example.

### Specifying Load Metadata

In addition to choosing a parser option, COPY supports other options to determine how to handle raw data. These options are considered load metadata, and you can specify metadata options in different parts of the COPY statement as follows:

<table>
<thead>
<tr>
<th>Metadata option</th>
<th>Column expression</th>
<th>COLUMN OPTION</th>
<th>FROM options</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELIMITER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ENCLOSED BY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ESCAPE AS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NULL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TRIM</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RECORD TERMINATOR</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>SKIP</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>SKIP BYTES</td>
<td></td>
<td>Y (fixed-width only)</td>
<td></td>
</tr>
<tr>
<td>TRAILING NULLCOLS</td>
<td></td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

The following precedence rules apply to all data loads:

- All column-level parameters override statement-level parameters.
- COPY uses the statement-level parameter if you do not specify a column-level parameter.
COPY uses the default metadata values for the DELIMITER, ENCLOSED BY, ESCAPE AS, and NULL options if you do not specify them at either the statement or column level.

When you specify any metadata options, COPY uses the parser to produce the best results and stores the raw data and its corresponding metadata in the following formats:

<table>
<thead>
<tr>
<th>Raw data format</th>
<th>Metadata format</th>
<th>Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTF-8</td>
<td>UTF-8</td>
<td>DELIMITER</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary</td>
<td>NATIVE</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Binary</td>
<td>NATIVE VARCHAR</td>
</tr>
<tr>
<td>UTF-8</td>
<td>UTF-8</td>
<td>FIXEDWIDTH</td>
</tr>
</tbody>
</table>

Interpreting Last Column End of Row Values

When bulk-loading delimited text data using the default parser (DELIMITED), the last column end of row value can be any of the following:

- Record terminator
- EOF designator
- Delimiter and a record terminator

**Note:** The FIXEDWIDTH parser always requires exactly a record terminator. No other permutations work.

For example, given a three-column table, the following input rows for a COPY statement using a comma (,) delimiter are each valid:

```
1,1,11
1,1,,
1,1,,
```

The following examples illustrate how COPY can interpret different last column end of data row values.

Using a Single End of Row Definition

To see how COPY interprets a single end of row definition:
1. Create a two-column table `two_col`, specifying column b with a default value of 5:

```sql
=> CREATE TABLE two_col (a int, b int DEFAULT 5);
CREATE TABLE
```

2. COPY the `two_col` table using a comma (,) delimiter, and enter values for only one column (as a single, multi-line entry):

```sql
=> COPY two_col from stdin delimiter ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1,
>> 1,
>> .
```

The COPY statement complete successfully.

3. Query table `two_col`, to display the two NULL values for column b as blank:

```sql
=> SELECT * FROM two_col;
   a | b
  ---+---
    1 |   
    1 |   
(2 rows)
```

Here, COPY expects two values for each column, but gets only one. Each input value is followed by a delimiter (,), and an implicit record terminator (a newline character, \n). You supply a record terminator with the ENTER or RETURN key. This character is not represented on the screen.

In this case, the delimiter (,) and record terminator (\n) are handled independently. COPY interprets the delimiter (,) to indicate the end of one value, and the record terminator (\n) to specify the end of the column row. Since no value follows the delimiter, COPY supplies an empty string before the record terminator. By default, the empty string signifies a NULL, which is a valid column value.

Using a Delimiter and Record Terminator End of Row Definition

To use a delimiter and record terminator together as an end of row definition:

1. Copy column a (a) of the `two_col` table, using a comma delimiter again, and enter two values:
The COPY statement again completes successfully.

2. Query table `two_col` to see that column b now includes two rows with its default value (5):

```
=> SELECT * FROM two_col;
a | b  
---+---
1 |   
1 |   
2 | 5  
2 | 5  
(4 rows)
```

In this example, COPY expects values for only one column, because of the column (a) directive. As such, COPY interprets the delimiter and record terminator together as a single, valid, last column end of row definition. Before parsing incoming data, COPY populates column b with its default value, because the table definition has two columns and the COPY statement supplies only one. This example populates the second column with its default column list value, while the previous example used the supplied input data.

### Loading Delimited Data

When bulk-loading data using the default (DELIMITED) parser, you can specify the following parameters at either a statement or column level:

- **ENCLOSED BY**
- **ESCAPE AS**
- **NULL**
- **DELIMITER**

Vertica assumes that data is in the UTF-8 encoding.
Loading Special Characters As Literals

The default COPY statement escape key is a backslash (\). By preceding any special character with an escape character, COPY interprets the character that follows literally, and copies it into the database. These are the special characters that you escape to load them as literals:

<table>
<thead>
<tr>
<th>Special Character</th>
<th>COPY Statement Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical bar (</td>
<td>)</td>
</tr>
<tr>
<td>Empty string (' ')</td>
<td>Default COPY ... NULL string.</td>
</tr>
<tr>
<td>Backslash ()</td>
<td>Default COPY ... ESC character.</td>
</tr>
<tr>
<td>Newline and other control characters</td>
<td>Various</td>
</tr>
</tbody>
</table>

To use a special character as a literal, prefix it with an escape character. For example, to include a literal backslash (\) in the loaded data (such as when including a file path), use two backslashes (\\). COPY removes the escape character from the input when it loads escaped characters.

Using a Custom Column Separator (DELIMITER)

The default COPY delimiter is a vertical bar (|). The DELIMITER is a single ASCII character used to separate columns within each record of a file. Between two delimiters, COPY interprets all string data in load files as characters. Do not enclose character strings in quotes, since quote characters are also treated as literals between delimiters.

You can define a different delimiter using any ASCII value in the range E'\000' to E'\177' inclusive. For instance, if you are loading CSV data files, and the files use a comma () character as a delimiter, you can change the default delimiter to a comma. You cannot use the same character for both the DELIMITER and NULL options.

If the delimiter character is among a string of data values, use the ESCAPE AS character (\ by default) to indicate that the delimiter should be treated as a literal.

The COPY statement accepts empty values (two consecutive delimiters) as valid input data for CHAR and VARCHAR data types. COPY stores empty columns as an empty string (' '). An empty string is not equivalent to a NULL string.

To indicate a non-printing delimiter character (such as a tab), specify the character in extended string syntax (E'...'). If your database has StandardConformingStrings enabled, use a
Unicode string literal (U'...'). For example, use either E'\t' or U'\009' to specify tab as the delimiter.

Using a Custom Column Option DELIMITER

This example, redefines the default delimiter through the COLUMN OPTION parameter.

1. Create a simple table.

```sql
=> CREATE TABLE t(pk INT, col1 VARCHAR(10), col2 VARCHAR(10), col3 VARCHAR(10), col4 TIMESTAMP);
```

2. Use the COLUMN OPTION parameter to change the col1 default delimiter to a tilde (~).

```sql
=> COPY t COLUMN OPTION(col1 DELIMITER '~') FROM STDIN NO COMMIT;
>> 1|ee~gg|yy|1999-12-12
>> \.
=> SELECT * FROM t;
<table>
<thead>
<tr>
<th>pk</th>
<th>col1</th>
<th>col2</th>
<th>col3</th>
<th>col4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ee</td>
<td>gg</td>
<td>yy</td>
<td>1999-12-12 00:00:00</td>
</tr>
</tbody>
</table>
```

Defining a Null Value (NULL)

The default NULL value for COPY is an empty string ('''). You can specify a NULL as any ASCII value in the range E'\001' to E'\177' inclusive (any ASCII character except NUL: E'\000'). You cannot use the same character for both the DELIMITER and NULL options.

When NULL is an empty string ('''), use quotes to insert an empty string instead of a NULL. For example, using NULL " ENCLOSED BY '':'':

- 1|3 — Inserts a NULL in the second column.
- 1|''|3 — Inserts an empty string instead of a NULL in the second columns.

To input an empty or literal string, use quotes (ENCLOSED BY); for example:

```sql
NULL "'null' literal"
```

A NULL is case-insensitive and must be the only value between the data field delimiters. For example, if the null string is NULL and the delimiter is the default vertical bar (|):

|NULL| indicates a null value.
NULL does not indicate a null value.

When you use the COPY command in a script, you must substitute a double-backslash for each null string that includes a backslash. For example, the scripts used to load the example database contain:

```
COPY ... NULL E'\n' ...
```

### Loading NULL Values

You can specify NULL by entering fields without content into a data file, using a field delimiter.

For example, given the default delimiter (|) and default NULL (empty string) definition, COPY inserts the following input data:

```
| | 1| 2 | 3
4 | 5
6 | |
```

into the table as follows:

```
(null, null, 1)(null, 2, 3)
(4, null, 5)
(6, null, null)
```

If NULL is set as a literal ('null'), COPY inserts the following inputs:

```
null | null | one | 2 | 3
4 | null | 5
6 | null | null
```

as follows:

```
(null, null, 1)(null, 2, 3)
(4, null, 5)
(6, null, null)
```

### Filling Columns with Trailing Nulls (TRAILING NULLCOLS)

Loading data using the TRAILING NULLCOLS option inserts NULL values into any columns without data. Before inserting TRAILING NULLCOLS, Vertica verifies that the column does not have a NOT NULL constraint.

To use the TRAILING NULLCOLS parameter to handle inserts with fewer values than data columns:
1. Create a table:

```sql
=> CREATE TABLE z (a INT,
                     b INT,
                     c INT);
```

2. Insert some values into the table:

```sql
=> INSERT INTO z VALUES (1, 2, 3);
```

3. Query table z to see the inputs:

```sql
=> SELECT * FROM z;
 a | b | c
---+---+---
 1 | 2 | 3
(1 row)
```

4. Insert two rows of data from STDIN, using TRAILING NULLCOLS:

```sql
=> COPY z FROM STDIN TRAILING NULLCOLS;
>> 4 | 5 | 6
>> 7 | 8
>> 
```

5. Query table z again to see the results. Using TRAILING NULLCOLS, the COPY statement correctly handled the third row of column c, which had no value:

```sql
=> SELECT * FROM z;
 a | b | c
---+---+---
 1 | 2 | 3
 4 | 5 | 6
 7 | 8 | 
(3 rows)
```

**Attempting to Fill a NOT NULL Column with TRAILING NULLCOLS**

You cannot use TRAILING NULLCOLS on a column that has a NOT NULL constraint. For instance:
1. Create a table `n`, declaring column `b` with a NOT NULL constraint:

```sql
=> CREATE TABLE n ( a INT,
    b INT NOT NULL,
    c INT );
```

2. Insert some table values:

```sql
=> INSERT INTO n VALUES (1, 2, 3);
=> SELECT * FROM n;

a | b | c
---+---+---
1 | 2 | 3
(1 row)
```

3. Use COPY with TRAILING NULLCOLS on table `n` to see the COPY error due to the column constraint:

```sql
=> COPY n FROM STDIN trailing nullcols abort on error;
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.

>> 4 | 5 | 6
>> 7 | 8
>> 9
>> .
ERROR:  COPY: Input record 3 has been rejected (Cannot set trailing column to NULL as column 2 (b) is NOT NULL)
```

4. Query the table to see that the COPY statement values were rejected:

```sql
=> SELECT * FROM n;

a | b | c
---+---+---
1 | 2 | 3
(1 row)
```

### Changing the Default Escape Character (ESCAPE AS)

The default escape character is a backslash (\). To change the default to a different character, use the `ESCAPE AS` option. To use an alternative escape character:

```sql
=> COPY mytable FROM '/data/input.txt' ESCAPE AS E('\\01');
```

You can set the escape character to be any ASCII value value in the range E '\'01' to E '\177' inclusive.
Eliminating Escape Character Handling

If you do not want any escape character and want to prevent any characters from being interpreted as escape sequences, use the NO ESCAPE option as part of the COPY statement.

Delimiting Characters (ENCLOSED BY)

The COPY ENCLOSED BY parameter lets you set an ASCII character to delimit characters to embed in string values. You can use any ASCII value in the range E'\01' to E'\177' inclusive (any ASCII character except NULL: E'\000') for the ENCLOSED BY value. Using double quotation marks (" ) is the most commonly used quotation character. For instance, the following parameter specifies that input data to the COPY statement is enclosed within double quotes:

ENCLOSED BY '"'

With the following input (using the default DELIMITER (| ) character), specifying:

"vertica | value"

Results in:

- Column 1 containing "vertica"
- Column 2 containing value"

Notice the double quotes (" ) before vertica and after value.

Using the following sample input data as follows, columns are distributed as shown:

"1", "vertica,value", ",", ""

Alternatively, write the above example using any ASCII character of your choosing:

~1~, ~vertica,value~, ~~, ~~

If you use a single quote ('), rather than double quotes (" ) as the ENCLOSED BY character, you must escape it using extended string syntax, a Unicode literal string (if StandardConformingStrings is enabled), or by using four single quotes:
Using any of the definitions means the following input is properly parsed:

'1', 'vertica,value', ',', '"'

See String Literals (Character) for an explanation of the string literal formats you can use to specify the ENCLOSED BY parameter.

Use the ESCAPE AS character to embed the ENCLOSED BY delimiter within character string values. For example, using the default ESCAPE AS character (\) and double quote as the ENCLOSED BY character, the following input returns "vertica":

"\"vertica\""

Using ENCLOSED BY for a Single Column

The following example uses double quotes to enclose a single column (rather than the entire row). The COPY statement also specifies a comma (,) as the delimiter.

```sql
=> COPY Retail.Dim (Dno, Dname ENCLOSED BY '"', Dstore) FROM '/home/dbadmin/dim3.txt'
DELMITER ','
EXCEPTIONS '/home/dbadmin/exp.txt';
```

This example correctly loads data such as:

123,"Smith, John",9832

Specifying a Custom End of Record String (RECORD TERMINATOR)

To specify the literal character string that indicates the end of a data file record, use the RECORD TERMINATOR parameter, followed by the string to use. If you do not specify a value, then Vertica attempts to determine the correct line ending, accepting either just a linefeed (E'\n') common on UNIX systems, or a carriage return and linefeed (E'\r\n') common on Windows platforms.

For example, if your file contains comma-separated values terminated by line feeds that you want to maintain, use the RECORD TERMINATOR option to specify an alternative value:

```sql
=> COPY mytable FROM STDIN DELIMITER ',' RECORD TERMINATOR E'\n';
```
To specify the RECORD TERMINATOR as non-printing characters, use either the extended string syntax or Unicode string literals. The following table lists some common record terminator characters. See String Literals for an explanation of the literal string formats.

<table>
<thead>
<tr>
<th>Extended String Syntax</th>
<th>Unicode Literal String</th>
<th>Description</th>
<th>ASCII Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>E'\b'</td>
<td>U&amp;'\0008'</td>
<td>Backspace</td>
<td>8</td>
</tr>
<tr>
<td>E'\t'</td>
<td>U&amp;'\0009'</td>
<td>Horizontal tab</td>
<td>9</td>
</tr>
<tr>
<td>E'\n'</td>
<td>U&amp;'\000a'</td>
<td>Linefeed</td>
<td>10</td>
</tr>
<tr>
<td>E'\f'</td>
<td>U&amp;'\000c'</td>
<td>Formfeed</td>
<td>12</td>
</tr>
<tr>
<td>E'\r'</td>
<td>U&amp;'\000d'</td>
<td>Carriage return</td>
<td>13</td>
</tr>
<tr>
<td>E'\'</td>
<td>U&amp;'\005c'</td>
<td>Backslash</td>
<td>92</td>
</tr>
</tbody>
</table>

If you use the RECORD TERMINATOR option to specify a custom value, be sure the input file matches the value. Otherwise, you may get inconsistent data loads.

**Note:** The record terminator cannot be the same as DELIMITER, NULL, ESCAPE, or ENCLOSED BY.

If using JDBC, Vertica recommends that you use the following value for the RECORD TERMINATOR:

```java
System.getProperty("line.separator")
```

### Examples

The following examples use a comma (,) as the DELIMITER for readability.

```
,1,2,3,,1,2,3
1;2,3,
```

Leading (,1) and trailing (3,) delimiters are ignored. Thus, the rows all have three columns.

<table>
<thead>
<tr>
<th>123,\n&quot;,\n&quot;,456</th>
<th>Using a non-default null string, the row is interpreted as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>123newline</td>
<td></td>
</tr>
<tr>
<td>\n</td>
<td></td>
</tr>
<tr>
<td>456</td>
<td></td>
</tr>
</tbody>
</table>

```
123,this\\, that\\, or the other,something else,456
```

The row would be interpreted as:

```
123this, that, or the other something else 456
```
Loading Binary (Native) Data

You can load binary data using the NATIVE parser option, except with COPY LOCAL, which does not support this option. Since binary-format data does not require the use and processing of delimiters, it precludes the need to convert integers, dates, and timestamps from text to their native storage format, and improves load performance over delimited data. All binary-format files must adhere to the formatting specifications described in Appendix: Creating Native Binary Format Files.

Native binary format data files are typically larger than their delimited text format counterparts, so compress the data before loading it. The NATIVE parser does not support concatenated compressed binary files. You can load native (binary) format files when developing plug-ins to ETL applications.

There is no copy format to load binary data byte-for-byte because the column and record separators in the data would have to be escaped. Binary data type values are padded and translated on input, and also in the functions, operators, and casts supported.

Loading Hexadecimal, Octal, and Bitstring Data

You can use the formats hexadecimal, octal, and bitstring only to load binary columns. To specify these column formats, use the COPY statement's FORMAT options:

- Hexadecimal
- Octal
- Bitstring

The following examples illustrate how to use the FORMAT option.

1. Create a table:

```sql
=> CREATE TABLE t(oct VARBINARY(5),
                 hex VARBINARY(5),
                 bitstring VARBINARY(5) );
```

2. Create the projection:

```sql
=> CREATE PROJECTION t_p(oct, hex, bitstring) AS SELECT * FROM t;
```
3. Use a COPY statement with the STDIN clause, specifying each of the formats:

```sql
=> COPY t (oct FORMAT 'octal', hex FORMAT 'hex',
    bitstring FORMAT 'bitstring')
    FROM STDIN DELIMITER ',';
```

4. Enter the data to load, ending the statement with a backslash (\) and a period (.) on a separate line:

```
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 141142143144145,0x6162636465,01100001011000100110001100110010001100101
>> .
```

5. Use a select query on table `t` to view the input values results:

```sql
=> SELECT * FROM t;
 oct | hex | bitstring
--------+-------+---------
abcde   | abcde | abcde   
(1 row)
```

COPY uses the same default format to load binary data, as used to input binary data. Since the backslash character (\) is the default escape character, you must escape octal input values. For example, enter the byte \141 as \\141.

**Note:** If you enter an escape character followed by an invalid octal digit or an escape character being escaped, COPY returns an error.

On input, COPY translates string data as follows:

- Uses the **HEX_TO_BINARY** function to translate from hexadecimal representation to binary.
- Uses the **BITSTRING_TO_BINARY** function to translate from bitstring representation to binary.

Both functions take a VARCHAR argument and return a VARBINARY value.

You can also use the escape character to represent the (decimal) byte 92 by escaping it twice; for example, '\\\141'. Note that vsql inputs the escaped backslash as four backslashes. Equivalent inputs are hex value '0x5c' and octal value '\134' (134 = 1 x 8^2 + 3 x 8^1 + 4 x 8^0 = 92).

You can load a delimiter value if you escape it with a backslash. For example, given delimiter ' | ', '\\001\\002' is loaded as {1, 124, 2}, which can also be represented in octal format as '\\001\\174\\002'.
If you insert a value with more bytes than fit into the target column, COPY returns an error. For example, if column c1 is VARBINARY(1):

```sql
=> INSERT INTO t (c1) values ('ab');  ERROR: 2-byte value too long for type Varbinary(1)
```

If you implicitly or explicitly cast a value with more bytes than fit the target data type, COPY silently truncates the data. For example:

```sql
=> SELECT 'abcd'::binary(2);
 binary
 --------
 ab
 (1 row)
```

**Hexadecimal Data**

The optional '0x' prefix indicates that a value is hexadecimal, not decimal, although not all hexadecimal values use A-F; for example, 5396. COPY ignores the 0x prefix when loading the input data.

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

**Octal Data**

Loading octal format data requires that each byte be represented by a three-digit octal code. The first digit must be in the range [0,3] and the second and third digits must both be in the range [0,7].

If the length of an octal value is not a multiple of three, or if one of the three digits is not in the proper range, the value is invalid and COPY rejects the row in which the value appears. If you supply an invalid octal value, COPY returns an error. For example:

```sql
SELECT '\000\387'::binary(0);
ERROR: invalid input syntax for type binary
```

Rows that contain binary values with invalid octal representations are also rejected. For example, COPY rejects '\008' because '\008' is not a valid octal number.

**BitString Data**

Loading bitstring data requires that each character must be zero (0) or one (1), in multiples of eight characters. If the bitstring value is not a multiple of eight characters, COPY treats the first
\( n \) characters as the low bits of the first byte (furthest to the left), where \( n \) is the remainder of the value's length, divided by eight.

**Examples**

The following example shows `VARBINARY HEX_TO_BINARY(VARCHAR)` and `VARCHAR TO_HEX(VARBINARY)` usage.

1. Create table \( t \) and and its projection with binary columns:

   ```
   => CREATE TABLE t (c BINARY(1));
   => CREATE PROJECTION t_p (c) AS SELECT c FROM t;
   ```

2. Insert minimum and maximum byte values, including an IP address represented as a character string:

   ```
   => INSERT INTO t values(HEX_TO_BINARY('0x00'));
   => INSERT INTO t values(HEX_TO_BINARY('0xFF'));
   => INSERT INTO t values(V6_ATON('2001:DB8::8:800:200C:417A'));
   ```

   Use the `TO_HEX` function to format binary values in hexadecimal on output:

   ```
   => SELECT TO_HEX(c) FROM t;
   to_hex
   --------
   00
   ff
   20
   (3 rows)
   ```

**See Also**

- Binary Data Types
- Formatting Functions
- ASCII

**Loading Native Varchar Data**

Use the `NATIVE VARCHAR` parser option when the raw data consists primarily of CHAR or VARCHAR data. `COPY` performs the conversion to the actual table data types on the database server. This parser option is not supported with `COPY LOCAL`. 
Using NATIVE VARCHAR does not provide the same efficiency as NATIVE. However, NATIVE VARCHAR precludes the need to use delimiters or to escape special characters, such as quotes, which can make working with client applications easier.

**Note:** NATIVE VARCHAR does not support concatenated compressed files.

Batch data inserts performed through the Vertica ODBC and JDBC drivers automatically use the NATIVE VARCHAR format.

### Loading Fixed-Width Format Data

Use the FIXEDWIDTH parser option to bulk load fixed-width data. You must specify the COLSIZES option values to specify the number of bytes for each column. The definition of the table you are loading (COPY table f (x, y, z)) determines the number of COLSIZES values to declare.

To load fixed-width data, use the COLSIZES option to specify the number of bytes for each input column. If any records do not have values, COPY inserts one or more null characters to equal the specified number of bytes. The last record in a fixed-width data file must include a record terminator to determine the end of the load data.

### Supported Options for Fixed-Width Data Loads

Loading fixed-width data supports the options listed in the [COPY Option Parser Dependencies](#). These options are not supported:

- **DELIMITER**
- **ENCLOSED BY**
- **ESCAPE AS**
- **TRAILING NULLCOLS**

### Using Nulls in Fixed-Width Data

The default NULL string for a fixed-width load cannot be an empty string, and instead, consists of all spaces. The number of spaces depends on the column width declared with the COLSIZES (integer, [...]) option.
For fixed-width loads, the NULL definition depends on whether you specify NULL at the column or statement level:

- **Statement level**: NULL must be defined as a single-character. The default (or custom) NULL character is repeated for the entire width of the column.

- **Column level**: NULL must be defined as a string whose length matches the column width.

For fixed-width loads, if the input data column has fewer values than the specified column size, COPY inserts NULL characters. The number of NULLs must match the declared column width. If you specify a NULL string at the column level, COPY matches the string with the column width.

**Note**: To turn off NULLs, use the NULL AS option and specify NULL AS ''. 

### Defining a Null Character (Statement Level)

1. Create a two-column table (fw):

   ```
   => CREATE TABLE fw(co int, ci int);
   CREATE TABLE
   ```

2. Copy the table, specifying null as 'N', and enter some data:

   ```
   => COPY fw FROM STDIN FIXEDWIDTH colsizes(2,2) null AS 'N' NO COMMIT;
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> NN12
   >> 23NN
   >> NNNN
   >> nnnn
   >> \
   ```

3. Select all (*) from the table:

   ```
   => SELECT * FROM fw;
   co | ci
   ------
   23 | |
   | |
   | 12
   (4 rows)
   ```
Defining a Custom Record Terminator

To define a record terminator other than the COPY default when loading fixed-width data, take these steps:

1. Create table `fw` with two columns, `co` and `ci`:

   ```sql
   \=> CREATE TABLE fw(co int, ci int);
   CREATE TABLE
   ```

2. Copy table `fw`, specifying two 2-byte column sizes, and specifying a comma (,) as the record terminator:

   ```sql
   \=> COPY fw FROM STDIN FIXEDWIDTH colsizes(2,2) RECORD TERMINATOR ',';
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> 1234,1444,6666
   >> \. 
   ```

3. Query all data in table `fw`:

   ```sql
   \=> SELECT * FROM fw;
   co | ci
   ----|----
   12 | 34
   14 | 44
   (2 rows)
   ```

   The SELECT output indicates only two values. COPY rejected the third value (6666) because it was not followed by a comma (,) record terminator. Fixed-width data requires a trailing record terminator only if you explicitly specify a record terminator explicitly.

Copying Fixed-Width Data

Use COPY FIXEDWIDTH COLSIZES (n [, ...]) to load files into a Vertica database. By default, all spaces are NULLs. For example:

```sql
\=> CREATE TABLE mytest(co int, ci int);
\=> CREATE PROJECTION mytest_p1 AS SELECT * FROM mytest SEGMENTED BY HASH(co) ALL NODES;
\=> COPY mytest(co,ci) FROM STDIN FIXEDWIDTH colsizes(6,4) NO COMMIT;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> \\
\=> SELECT * FROM mytest ORDER BY co;
```
Skipping Content in Fixed-Width Data

The COPY statement has two options to skip input data. The SKIP BYTES option is only for fixed-width data loads:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIP BYTES num-bytes</td>
<td>Skips the specified number of bytes from the input data.</td>
</tr>
<tr>
<td>SKIP num-records</td>
<td>Skips the specified number of records.</td>
</tr>
</tbody>
</table>

The following example uses SKIP BYTES to skip 11 bytes when loading a fixed-width table with two columns (4 and 6 bytes):

1. Copy a table using SKIP BYTES:

```sql
=> COPY fw FROM STDIN FIXEDWIDTH colsizes (4,6) SKIP BYTES 11;
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 2222666666  
>> 1111999999  
>> 1632641282  
>> \.
```

2. Query all data in table fw:

```sql
=> SELECT * FROM fw ORDER BY co;  
co   |  ci  
-----|------
1111 | 999999  
1632 | 641282  
(2 rows)
```

The output confirms that COPY skipped the first 11 bytes of loaded data.

The following example uses SKIP when loading a fixed-width (4,6) table:

1. Copy a table, using SKIP to skip two records of input data:

```sql
=> COPY fw FROM STDIN FIXEDWIDTH colsizes (4,6) SKIP 2;  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 2222666666  
>> 1111999999
```
2. Query all data in table fw:

```sql
=> SELECT * FROM fw ORDER BY co;
  co | ci
-----------
1632 | 641282
3333 | 888888
(2 rows)
```

The output confirms that COPY skipped the first two records of load data.

### Trimming Characters in Fixed-Width Data Loads

Use the TRIM option to trim a character. TRIM accepts a single-byte character, which is trimmed at the beginning and end of the data. For fixed-width data loads, when you specify a TRIM character, COPY first checks to see if the row is NULL. If the row is not null, COPY trims the character(s). The next example instructs COPY to trim the character A, and shows the results:

1. Copy table fw, specifying TRIM character A:

```sql
=> COPY fw FROM STDIN FIXEDWIDTH colsizes(4,6) TRIM 'A';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> A22AA444444
>> A22AA4444A
>> \.
```

2. Query all data in table fw:

```sql
=> SELECT * FROM fw ORDER BY co;
  co | ci
-------
 22 | 4444
 22 | 444444
(2 rows)
```

### Using Padding in Fixed-Width Data Loads

By default, the padding character is ' ' (a single space). The padding behavior for fixed-width data loads is similar to how a space is treated in other formats, differing by data type as
follows:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Leading and trailing spaces</td>
</tr>
<tr>
<td>Bool</td>
<td>Leading and trailing spaces</td>
</tr>
<tr>
<td>Float</td>
<td>Leading and trailing spaces</td>
</tr>
<tr>
<td>[var]Binary</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>[Var]Char</td>
<td>Trailing spaces if string is too large</td>
</tr>
<tr>
<td>DateInterval</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>Time</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>TimestampTZ</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>TimeTZ</td>
<td>None, all characters are significant.</td>
</tr>
<tr>
<td>Date (formatted)</td>
<td>Use the COPY FORMAT option string to match the expected column length.</td>
</tr>
<tr>
<td>Numerics</td>
<td>Leading and trailing spaces</td>
</tr>
</tbody>
</table>

**Bulk Loading and Exporting Data From Amazon S3**

The Vertica library for Amazon Web Services (AWS) is a set of functions and configurable session parameters. These parameters allow you to directly exchange data between Vertica and Amazon S3 storage without any third-party scripts or programs.

To use the AWS library, you must have access to an Amazon S3 storage account.

To load data from S3 in text, delimited, or Parquet format, you can use COPY FROM instead of using this library. See [Specifying COPY FROM Options](#) for more information.

**Related Topics**

- Configure the AWS Library
- Loading Data From Amazon S3
Configuring the Vertica Library for Amazon Web Services

Configure the Vertica library for Amazon Web Services (AWS) by setting session parameters with your AWS access key credentials and region. You can set your session parameters directly, or you can store your credentials in a table and set them with the AWS_SET_CONFIG function.

Because the AWS library is configured with session parameters, you must reconfigure the library with each new session.

**Important:** Your AWS access key ID and secret access key are different from your account access credentials. For more information about AWS access keys, visit the AWS documentation.

Use either of the following methods to securely set and store your AWS account credentials:

- **Configuring Session Parameters Directly**
- **Configuring Session Parameters Using Keychains**

**Note:** To increase security, configure session parameters directly to avoid storing credentials within Vertica.

**AWS Access Key Requirements**

In order to communicate with AWS, your access key must have the following permissions:

- s3:GetObject
- s3:PutObject
- s3:ListBucket

For security purposes, Vertica recommends that you create a separate access key with limited permissions specifically for use with the Vertica Library for AWS.
Configuring Session Parameters Directly

Set the following session parameters for AWS using your own credentials:

- **aws_id** — This value is your AWS access key ID.
  
  ```sql
  => ALTER SESSION SET UDPARAMETER FOR awslib aws_id='AKABC0EXAMPLEPKXY2Q';
  ```

- **aws_secret** — This value is your AWS secret access key.
  
  ```sql
  => ALTER SESSION SET UDPARAMETER FOR awslib aws_secret='CEXAMPLE3tEXAMPLE1wEXAMPLEFrFEXAMPLE6+Yz';
  ```

- **aws_region** - This value is the AWS region associated with the S3 bucket you intend to access. Left unconfigured, aws_region will default to us-east-1. It identifies the default server used by Amazon S3.
  
  ```sql
  => ALTER SESSION SET UDPARAMETER FOR awslib aws_region='us-east-1';
  ```

**Important:** Parameter values are case sensitive.

### Configuring Session Parameters Using Credentials Stored in a Table

You can place your credentials in a table and secure them with a row-level access policy. You can then call your credentials with the AWS_SET_CONFIG scalar meta-function. This approach allows you to store your credentials on your cluster for future session parameter configuration. You must have dbadmin access to create access policies.

1. Create a table with rows or columns corresponding with your credentials:

   ```sql
   => CREATE TABLE keychain(accesskey varchar, secretaccesskey varchar);
   ```

2. Store your credentials in the corresponding columns:

   ```sql
   => COPY keychain FROM STDIN;
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   => AEXAMPLEI5EXAMPLEYXQ|CCEXAMPLEtFjTEXAMPLEiEXAMPLE6+Yz
   => \.
   ```

3. Set a row-level access policy appropriate to your security situation.
4. With each new session, configure your session parameters by calling the AWS_SET_CONFIG parameter in a SELECT statement:

```sql
=> SELECT AWS_SET_CONFIG('aws_id', accesskey), AWS_SET_CONFIG('aws_secret', secretaccesskey)
FROM keychain;
aws_set_config | aws_set_config
-----------------+-----------------
aws_id           | aws_secret      
(1 row)
```

**Important:** Vertica recommends that you *not* use the AWS_SET_CONFIG function to configure your library directly. When you do so, your credentials are recorded in the server log.

5. After you have configured your session parameters, verify them:

```sql
=> SHOW SESSION UDPARAMETER ALL;
```

Related Topics

- Import Data from Amazon S3 Using the Vertica AWS Library
- Export Data to Amazon S3 Using the Vertica AWS Library
- AWS_SET_CONFIG
- AWS_GET_CONFIG

Loading Data From Amazon S3

After you configure the Vertica library for Amazon Web Services (AWS), you can copy data from S3. To do so, use COPY with the S3 UDSource containing the location of your S3 bucket and object. You can use either a standard HTTPS URL, or the S3 URL scheme, as the following examples show.

**Use COPY with a standard HTTPS URL:**

```sql
=> COPY exampleTable SOURCE S3(url='https://s3.amazonaws.com/exampleBucket/object');
```

**Use COPY with the S3 URL scheme:**

```sql
=> COPY exampleTable SOURCE S3(url='s3://exampleBucket/object');
```
You can use the S3 UDSource with any UDParse or UDFilter to import any data format supported by Vertica. If Vertica encounters an error with AWS during an S3 import operation which it cannot resolve, it will abort the import and pass the AWS error information on in the Vertica error report.

Importing Multiple Specific Files

Import multiple specific files by separating the URLs with a bar:

```sql
=> COPY exampleTable SOURCE s3(url='s3://exampleBucket/object1|s3://exampleBucket/object2');
```

Specify your own delimiter:

```sql
=> COPY exampleTable SOURCE S3(url='s3://exampleBucket/object1,s3://exampleBucket/object2', delimiter=',');
```

Importing Multiple Files Using Glob Expansion

In addition to importing single and multiple files by specifying exact URL addresses and the URL parameter, you can use glob expansion to import multiple files in a bucket or subdirectory by specifying the location with the bucket parameter.

Import from all files in a bucket using glob expansion:

```sql
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/*');
```

Import from all files in a subdirectory using glob expansion:

```sql
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/subdirectory/*');
```

Import all files with a 'db_' prefix:

```sql
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/db_*');
```

Import all files with a .csv suffix:

```sql
=> COPY exampleTable SOURCE S3(bucket='s3://exampleBucket/*.csv');
```

Related Topics

- [Configuring the Vertica AWS Library](#)
- [Exporting Data to Amazon S3 From Vertica](#)
Exporting Data to Amazon S3 From Vertica

After you configure the library for Amazon Web Services (AWS), you can export Vertica data to Amazon S3 by calling the S3EXPORT() transform function. S3EXPORT() writes data to files, based on the URL you provide. Vertica performs all communication over HTTPS, regardless of the URL type you use. Vertica does not support virtual host style URLs. If you use HTTPS URL constructions, you must use path style URLs.

Note: If your S3 bucket contains a period in its path, then you should set the prepend_hash parameter to True.

You can control the output of S3EXPORT() in the following ways:

- Adjust the base query provided to S3EXPORT
- Adjust the parameters of S3EXPORT
- Adjust the partition of your result set with the OVER() clause

Adjust the Query Provided to S3EXPORT

By adjusting the query given to S3EXPORT(), you can export anything from tables to reporting queries.

This example exports a whole table:

```sql
=> SELECT S3EXPORT( * USING PARAMETERS url='s3://exampleBucket/object') OVER(PARTITION BEST) FROM exampleTable;
rows
-------
606
(1 row)
```

This example exports the results of a query:

```sql
=> SELECT S3EXPORT(customer_name, annual_income USING PARAMETERS url='s3://exampleBucket/object') OVER()
FROM public.customer_dimension
WHERE (customer_gender, annual_income) IN
(SELECT customer_gender, MAX(annual_income)
FROM public.customer_dimension
GROUP BY customer_gender);
```
Adjust the Parameters of S3EXPORT

Vertica exports partition data in chunks set by the chunksize parameter. In its default setting, this multipart parameter instructs AWS to recombine the chunks into a single S3 object. However, if your export exceeds the S3 object size limit, or you want to fix the size of output objects, you can disable multipart uploads. Disabling allows you to set chunking to any size you prefer under the AWS PUT limit. This approach causes each partition to write your export into multiple incrementing S3 objects.

This example shows how you can export a 27 MB file with multipart disabled and chunk size left to the 10 MB default:

```
=> SELECT S3EXPORT(name, company USING PARAMETERS url='s3://examplebucket/object',
               multipart=false) OVER()
                  FROM exampleTable;
```

Results:

```
10244k object.699b7f63.1
10244k object.699b7f63.2
6152k object.699b7f63.3
```

The base name of the resulting objects matches the provided URL. However, the partition is now split into three parts with a maximum chunk size of 10 MB.

Adjust the Partition of Your Result Set with the OVER Clause

Use the OVER clause to control your export partitions. Using the OVER() clause without qualification results in a single partition processed by the initiator for all of the query data. This example shows how to call the function with an unqualified OVER() clause:

```
=> SELECT S3EXPORT(name, company USING PARAMETERS url='s3://exampleBucket/object',
                delimiter=',') OVER()
                   FROM exampleTable WHERE company='Vertica';
```

Results:

```
10
```
You can also use window clauses, such as window partition clauses and window order clauses, to manage exported objects.

This example shows how you can use a window partition clause to partition S3 objects based on company values:

```sql
=> SELECT S3EXPORT(name, company
    USING PARAMETERS url='s3://exampleBucket/object',
    delimiter=','
) OVER(PARTITION BY company) AS MEDIAN
FROM exampleTable;
```

Related Topics

- Configure the AWS Library
- Loading Data From Amazon S3
- s3export Function

**Manipulating Source Data Columns**

When loading data, your source data might contain one or more columns that do not exist in the target table. Or, the source and target tables have matched columns, but you want to omit one or more source columns from the target table.

The **COPY** command's FILLER parameter helps you accomplish these tasks:

- Ignore input columns from the source data.
- Transform source data before loading it into the target table.

The FILLER parameter identifies a column of source data that the COPY command can ignore, or use to compute new values that are loaded into the target table. Define the FILLER parameter data type so it is compatible with the source data. For example, be sure to define a VARCHAR so its length can contain all source data; otherwise, data could be truncated. You can specify multiple filler columns, where each filler column is specified by its own FILLER parameter.
Use FILLER to Ignore Some Values and Compute New Values for the Target Table

Using SQL operators or functions, you can combine two or more source columns into one column; you can also split one column into multiple columns—for example, split date strings into day, month, and year components and load these into target table columns.

The following COPY statement reads all of the source data, but only loads the source columns first_name and last_name. It constructs the data for full_name by concatenating each of the source data columns. To do this, use the FILLER parameter to ignore the middle_name column on load, but use the column when concatenating data to populate the full_name column.

```sql
=> CREATE TABLE names(first_name VARCHAR(20), last_name VARCHAR(20), full_name VARCHAR(60));
CREATE TABLE
=> COPY names(first_name,
   middle_name FILLER VARCHAR(20),
   last_name,
   full_name AS first_name||'|'||middle_name||'|'||last_name)
   FROM STDIN;
```

Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

```text
>> Marc|Gregory|Smith
>> Sue|Lucia|Temp
>> Jon|Pete|Hamilton
>> .
```

```sql
=> SELECT * FROM names;
```

<table>
<thead>
<tr>
<th>first_name</th>
<th>last_name</th>
<th>full_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jon</td>
<td>Hamilton</td>
<td>Jon Pete Hamilton</td>
</tr>
<tr>
<td>Marc</td>
<td>Smith</td>
<td>Marc Gregory Smith</td>
</tr>
<tr>
<td>Sue</td>
<td>Temp</td>
<td>Sue Lucia Temp</td>
</tr>
</tbody>
</table>

(3 rows)

If the source field's data type is VARCHAR, be sure to set the VARCHAR length to ensure that the combined length of all FILLER source fields does not exceed the target column's defined length; otherwise, the COPY command could return with an error.

Using Parallel Load Streams

Vertica can divide the work of loading data, taking advantage of parallelism to speed up the operation. Vertica supports several types of parallelism:
Distributed load: Files in a multi-file load are loaded on several nodes in parallel, instead of all being loaded on a single node.

Apportioned load: A single large file or other single source is divided into segments (portions), which are assigned to several nodes to be loaded in parallel.

Caution: If the file is different on two nodes, an incorrect or incomplete result is returned, with no error or warning.

Apportioned load is enabled by default. If you want to disable it, set the EnableApportionLoad configuration parameter to 0.

Cooperative parse: A source being loaded on a single node uses multi-threading to parallelize the parse. Cooperative parse is enabled by default. If you want to disable it, set the EnableCooperativeParse configuration parameter to 0.

See General Parameters for information about the configuration parameters.

Specifying Distributed File Loads

The COPY FROM statement provides several ways to distribute a file load.

You can direct individual files in a multi-file load to specific nodes, as in the following example of distributed load.

```sql
=> COPY t FROM '/data/file1.dat' ON v_vmart_node0001, '/data/file2.dat' ON v_vmart_node0002;
```

You can use globbing (wildcard expansion) to specify a group of files with the ON ANY NODE directive, as in the following example.

- If apportioned load is enabled (the default), Vertica assigns different files to different nodes. Both the EnableApportionedLoad and EnableApportionedFileLoad must be set to 1.

- If apportioned load is disabled, a single node loads all the data.

```sql
=> COPY t FROM '/data/*.dat' ON ANY NODE;
```

If you have a single file instead of a group of files, you can still, potentially, benefit from apportioned load. The file must be large enough to divide into portions at least equal to ApportionedFileMinimumPortionSizeKB in size. You must also use a parser that supports apportioned load. The delimited parser built into Vertica supports apportioned load, but other parsers might not.
The following example shows how you can load a single large file using multiple nodes.

```sql
=> COPY t FROM '/data/bigfile.dat' ON ANY NODE;
```

You can limit the nodes that participate in an apportioned load. Doing so is useful if you need to balance several concurrent loads. Vertica apportions each load individually; it does not account for other loads that might be in progress on those nodes. You can, therefore, potentially speed up your loads by managing apportioning yourself.

The following example shows how you can apportion loads on specific nodes.

```sql
=> COPY t FROM '/data/big1.dat' (v_vmart_node0001, v_vmart_node0002, v_vmart_node0003),
    '/data/big2.dat' (v_vmart_node0004, v_vmart_node0005);
```

Loaded files can be of different formats, such as BZIP, GZIP, and others. However, because file compression is a filter, you cannot use apportioned load for a compressed file.

### Specifying Distributed Loads with Sources

You can also apportion loads using COPY WITH SOURCE. You can create sources and parsers with the User-Defined Load (UDL) API. If both the source and parser support apportioned load, and EnableApportionLoad is set, then Vertica attempts to divide the load among nodes.

The following example shows a load that you could apportion.

```sql
=> COPY t WITH SOURCE MySource() PARSER MyParser();
```

The built-in delimited parser supports apportioning, so you can use it with a user-defined source, as in the following example.

```sql
=> COPY t WITH SOURCE MySource();
```

### Number of Load Streams

Although the number of files you can load is not restricted, the optimal number of load streams depends on several factors, including:

- Number of nodes
- Physical and logical schemas
- Host processors
- Memory
- Disk space

Using too many load streams can deplete or reduce system memory required for optimal query processing. See Best Practices for Managing Workload Resources for advice on configuring load streams.

**Monitoring COPY Loads and Metrics**

You can check COPY loads using:

- Vertica functions
- LOAD_STREAMS system table
- LOAD_SOURCES system table

**Using Vertica Functions**

Two meta-functions return COPY metrics for the number of accepted or rejected rows from a COPY statement:

1. To get the number of accepted rows, use the `GET_NUM_ACCEPTED_ROWS` function:

   ```sql
   => select get_num_accepted_rows();
   get_num_accepted_rows
   ------------------------
   11
   (1 row)
   ```

2. To check the number of rejected rows, use the `GET_NUM_REJECTED_ROWS` function:

   ```sql
   => select get_num_rejected_rows();
   get_num_rejected_rows
   ------------------------
   0
   (1 row)
   ```

**Note:** `GET_NUM_ACCEPTED_ROWS` and `GET_NUM_REJECTED_ROWS` support loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use
these functions for multi-node loads.

**Using the CURRENT_LOAD_SOURCE Function**

You can include the `CURRENT_LOAD_SOURCE` function as a part of the `COPY` statement. Doing so allows you to insert into a column the input file name or value computed by this function.

To insert the file names into a column from multiple source files:

```sql
=> COPY t (c1, c2, c3 as CURRENT_LOAD_SOURCE()) FROM '/home/load_file_1' ON exampledb_node02,
    '/home/load_file_2' ON exampledb_node03 DELIMITER ',';
```

**Using the LOAD_STREAMS System Table**

Vertica includes a set of system tables that include monitoring information, as described in [Using System Tables](#). The `LOAD_STREAMS` system table includes information about load stream metrics from `COPY` and `COPY FROM VERTICA` statements. Thus, you can query table values to get `COPY` metrics.

To see all table columns:

```sql
=> SELECT * FROM load_streams;
```

**Using the STREAM NAME Parameter**

Using the `STREAM NAME` parameter as part of the `COPY` statement labels `COPY` streams explicitly so they are easier to identify in the `LOAD_STREAMS` system table.

To use the `STREAM NAME` parameter:

```sql
=> COPY mytable FROM myfile DELIMITER '|' DIRECT STREAM NAME 'My stream name';
```

The `LOAD_STREAMS` system table includes stream names for every `COPY` statement that takes more than 1-second to run. The 1-second duration includes the time to plan and execute the statement.

Vertica maintains system table metrics until they reach a designated size quota (in kilobytes). This quota is set through internal processes, which you cannot set or view directly.
Other LOAD_STREAMS Columns for COPY Metrics

These LOAD_STREAMS system table column values depend on the load status:

- ACCEPTED_ROW_COUNT
- REJECTED_ROW_COUNT
- PARSE_COMPLETE_PERCENT
- SORT_COMPLETE_PERCENT

When a COPY statement using the DIRECT option is in progress, the ACCEPTED_ROW_COUNT value can increase during parsing. This value can reach the maximum number of rows in the input file.

If COPY reads input data from multiple named pipes, the PARSE_COMPLETE_PERCENT value remains at zero (0) until all named pipes return an EOF. While COPY awaits an EOF from multiple pipes, it can appear to be hung. However, before canceling the COPY statement, check your system CPU and disk accesses to determine if any activity is in progress.

In a typical load, the PARSE_COMPLETE_PERCENT value can either increase slowly or jump quickly to 100%, if you are loading from named pipes or STDIN. However, SORT_COMPLETE_PERCENT remains at 0 when loading from named pipes or STDIN. After PARSE_COMPLETE_PERCENT reaches 100%, SORT_COMPLETE_PERCENT increases to 100%. Depending on the data sizes, a significant lag can occur between the time PARSE_COMPLETE_PERCENT reaches 100% and the time SORT_COMPLETE_PERCENT begins to increase.

This example shows how you can set the VSQL expanded display and then select various columns of data from the LOAD_STREAMS system table:

```sql
=> \pset expanded
Expanded display is on.
=> SELECT stream_name, table_name, load_start, accepted_row_count, rejected_row_count, read_bytes, unsorted_row_count, sorted_row_count, parse_complete_percent FROM load_streams;

<table>
<thead>
<tr>
<th>stream_name</th>
<th>fact-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>fact</td>
</tr>
<tr>
<td>load_start</td>
<td>2010-12-28 15:07:41.132053</td>
</tr>
<tr>
<td>accepted_row_count</td>
<td>900</td>
</tr>
<tr>
<td>rejected_row_count</td>
<td>100</td>
</tr>
<tr>
<td>read_bytes</td>
<td>11975</td>
</tr>
<tr>
<td>input_file_size_bytes</td>
<td>0</td>
</tr>
<tr>
<td>parse_complete_percent</td>
<td>0</td>
</tr>
<tr>
<td>unsorted_row_count</td>
<td>3600</td>
</tr>
<tr>
<td>sorted_row_count</td>
<td>3600</td>
</tr>
</tbody>
</table>
```
Using the LOAD_SOURCES System Table

The LOAD_STREAMS table shows the total number of rows that were loaded or rejected. Grouping this information by source can help you determine from where data is coming. The LOAD_SOURCES system table includes some of the same data as LOAD_STREAMS does but adds this source-specific information. If apportioning is enabled, LOAD_SOURCES also provides information about how loads are apportioned among nodes.

You can use this table to identify causes of differing query results. For example, you can use the following statement to create an external table based on globs:

```sql
=> CREATE EXTERNAL TABLE tt AS COPY WITH SOURCE AWS(dir = 'foo', file = '*');
```

If you select from this table, Vertica loads data from every file in the `foo` directory and creates one row in the LOAD_SOURCES table for each file. Suppose you later repeat the query and see different results. You could look at the LOAD_SOURCES table and discover that—between the two queries—somebody added another file to the `foo` directory. Because each file is recorded in LOAD_SOURCES, you can see the new file that explains the changed query results.

If you are using many data sources, you might prefer to disable this reporting. To disable reporting, set the LoadSourceStatisticsLimit configuration parameter to 0. This parameter sets the upper bound on the number of sources profiled by LOAD_SOURCES per load. The default value is 256.

Capturing Load Rejections and Exceptions

Loading data with COPY has two main phases, parsing and loading. During parsing, if COPY encounters errors it rejects the faulty data and continues loading data. Rejected data is created whenever COPY cannot parse a row of data. Following are some parser errors that can cause a rejected row:

- Unsupported parser options
- Incorrect data types for the table into which data is being loaded
- Malformed context for the parser in use
- Missing delimiters

Optionally, COPY can reject data and continue loading when transforming data during the load phase. This behavior is controlled by a configuration parameter. By default, COPY aborts a load if it encounters errors during the loading phase.

Several optional parameters let you determine how strictly COPY handles rejections. For example, you can have COPY fail when it rejects a single row, or allow a specific number of rejections before the load fails. This section presents the parameters to determine how COPY handles rejected data.

**Save Rejected Rows (REJECTED DATA and EXCEPTIONS)**

The COPY statement automatically saves a copy of each rejected row in a rejected-data file. COPY also saves a corresponding explanation of what caused the rejection in an exceptions file. By default, Vertica saves both files in a database catalog subdirectory, called CopyErrorLogs, as shown in this example:

```
  v_mart_node003_catalog\CopyErrorLogs\trans-STDIN-copy-from-rejected-data.1
  v_mart_node003_catalog\CopyErrorLogs\trans-STDIN-copy-from-exceptions.1
```

You can optionally save COPY rejections and exceptions in one of two other ways:

- Use the REJECTED DATA `reject_path` and EXCEPTIONS `except_path` parameters to save both outputs to locations of your choice. REJECTED DATA records rejected rows, while EXCEPTIONS records a description of why each row was rejected. If a path value is an existing directory or ends in '/', or the load includes multiple sources, files are written in that directory. (COPY creates the directory if it does not exist.) If a path value is a file, COPY uses it as a file prefix if multiple files are written.

- Use the REJECTED DATA AS TABLE `reject_table` clause. This option writes both the rejected data and the exception descriptions to the same table. For more information, see Saving Rejected Data To a Table.

**Note:** Vertica recommends saving rejected data to a table. However, saving to a table excludes saving to a default or specific rejected data file.

If you save rejected data to a table, the table files are stored in the data subdirectory. For example, in a VMart database installation, rejected data table records are stored in the RejectionTableData directory as follows:
COPY LOCAL Rejected Data

For COPY LOCAL operations, you must use the REJECTED DATA reject_path and EXCEPTIONS except_path parameters explicitly. The reject_path and except_path paths must reside on the client. If a path resolves to a storage location and the user invoking COPY is not a superuser, the following permissions are required:

- The storage location must have been created with the USER usage type (see CREATE LOCATION).
- The user must already have been granted access to the storage location where the files exist, as described in GRANT (Storage Location)

Enforce Truncating or Rejecting Rows (ENFORCELENGTH)

When parsing data of type CHAR, VARCHAR, BINARY, or VARBINARY, rows may exceed the target table length. By default, COPY truncates such rows without rejecting them.

Use the ENFORCELENGTH parameter to reject rows that exceed the target table.

For example, loading 'abc' into a table column specified as VARCHAR(2) results in COPY truncating the value to 'ab' and loading it. Loading the same row with the ENFORCELENGTH parameter causes COPY to reject the row.

Note: Vertica supports NATIVE and NATIVE VARCHAR values up to 65K. If any value exceeds this limit, COPY rejects the row, even when ENFORCELENGTH is not in use.

Specify a Maximum Number of Rejections (REJECTMAX)

The REJECTMAX parameter specifies the maximum number of logical records that can be rejected before a load fails. A rejected row consists of the data that could not be parsed (or optionally transformed) into the corresponding data type during a bulk load. Rejected data...
does not indicate referential constraints. For information about using constraints, and the option of enforcing constraints during bulk loading, see About Constraints.

When the number of rejected records becomes equal to the REJECTMAX value, the load fails. If you do not specify a value for REJECTMAX, or if the value is 0, COPY allows an unlimited number of exceptions to occur.

If you allow COPY to reject rows and proceed when it encounters transformation errors, consider using REJECTMAX to limit the impact. See Handling Transformation Errors.

Handling Transformation Errors

By default, COPY aborts a load if it encounters errors when performing transformations. This is the default because rejecting transformation errors is potentially more expensive than rejecting parse errors. Sometimes, however, you would prefer to load the data anyway and reject the problematic rows, the way it does for parse errors.

To have COPY treat errors in transformation expressions like parse errors, set the CopyFaultTolerantExpressions configuration parameter to 1. (See General Parameters.) Rows that are rejected during transformation, in the expression-evaluation phase of a data load, are written to the same destination as rows rejected during parsing. Use REJECTED DATA or REJECTED DATA AS TABLE to specify the output location.

You might want to enable transformation rejections if your data contains a few bad rows. By enabling these rejections, you can load the majority of your data and proceed. Vertica recommends using REJECTMAX when enabling transformation rejections.

If your data contains many bad values, then the performance for loading the good rows could be worse than with parser errors.

Abort Data Loads for Any Error (ABORT ON ERROR)

Using the ABORT ON ERROR argument is the most restrictive way to load data, because no exceptions or rejections are allowed. A COPY operation stops if any row is rejected. No data is loaded and Vertica rolls back the command.

If you use the ABORT ON ERROR as part of a CREATE EXTERNAL TABLE AS COPY FROM statement, the option is used whenever a query references the external table. The offending error is saved in the COPY exceptions or rejected data file.
Understanding Row Rejections and Rollback Errors

Depending on the type of error that COPY encounters, Vertica does one of the following:

- Rejects the offending row and loads other rows into a table
- Rolls back the entire COPY statement without loading any data

**Note:** If you specify ABORT ON ERROR with the COPY statement, the load automatically rolls back if COPY cannot parse any row.

The following table summarizes the reasons for rejected rows or rollbacks.

<table>
<thead>
<tr>
<th>Rejected Rows</th>
<th>Load Rollback</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY cannot parse rows that contain any of the following:</td>
<td>COPY rolls back a load if it encounters any of these conditions:</td>
</tr>
<tr>
<td>• Incompatible data types</td>
<td>• Server-side errors, such as lack of memory</td>
</tr>
<tr>
<td>• Missing fields</td>
<td>• Primary key or foreign key constraint violations</td>
</tr>
<tr>
<td>• Missing delimiters</td>
<td>• Loading NULL data into a NOT NULL column</td>
</tr>
<tr>
<td></td>
<td>• Transformation errors (by default)</td>
</tr>
</tbody>
</table>

This example illustrates what happens when Vertica cannot coerce a row to the requested data type. For example, in the following COPY statement, "a::INT + b::INT" is a SQL expression in which a and b are derived values:

```sql
=> CREATE TABLE t (i INT);
=> COPY t (a FILLER VARCHAR, b FILLER VARCHAR, i AS a::INT + b::INT) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> cat|dog
>> .
```

Vertica cannot parse the row to the requested data type and rejects the row:

```
ERROR 2827: Could not convert "cat" from column "FILLER*.a" to an int8
```
If a resolved to 'cat' and b to 'dog', the next expression 'cat'::INT + 'dog'::INT would return an expression evaluator error:

```sql
=> SELECT 'cat'::INT + 'dog'::INT;
ERROR 3681: Invalid input syntax for integer: "cat"
```

The following COPY statement would also roll back because Vertica cannot parse the row to the requested data type:

```sql
=> COPY t (a FILLER VARCHAR, i AS a::INT) FROM STDIN;
```

In the following COPY statement, Vertica rejects only the offending row without rolling back the statement. Instead of evaluating the 'cat' row as a VARCHAR type, COPY parses 'cat' directly as an INTEGER.

```sql
=> COPY t (a FILLER INT, i AS a) FROM STDIN;
```

In the following example, transformation errors are rejected instead of aborting the load.

```sql
=> ALTER DATABASE myDB SET CopyFaultTolerantExpressions = 1;
ALTER DATABASE

=> CREATE TABLE sales (price INTEGER);
COPY sales FROM STDIN REJECTED DATA AS TABLE sales_rej;
dollars 
.

=> SELECT rejected_data, rejected_reason FROM sales_rej;
rejected_data | rejected_reason
--------------+----------------------------------------
dollars       | Invalid integer format 'dollars' for column 1 (price) (1 row)
```

See Also

- Cast Failures

Saving Load Rejections (REJECTED DATA)

COPY load rejections are data rows that did not load due to a parser exception or, optionally, transformation error. By default, if you do not specify a rejected data file, COPY saves rejected data files to this location:

`catalog_dir/CopyErrorLogs/target_table-source-copy-from-rejected-data.n`

<table>
<thead>
<tr>
<th><code>catalog_dir/</code></th>
<th>The database catalog files directory, for example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>target_table</td>
<td>The table into which data was loaded (target_table).</td>
</tr>
<tr>
<td>source</td>
<td>The source of the load data, which can be STDIN, or a file name, such as baseball.csv.</td>
</tr>
<tr>
<td>copy-from-rejected-data.(n)</td>
<td>The default name for a rejected data file, followed by (n) suffix, indicating the number of files, such as .1, .2, .3. For example, this default file name indicates file 3 after loading from STDIN: fw-STDIN-copy-from-rejected-data.3.</td>
</tr>
</tbody>
</table>

Saving rejected data to the default location, or to a location of your choice, lets you review the file contents, resolve problems, and reload the data from the rejected data files. Saving rejected data to a table, lets you query the table to see rejected data rows and the reasons (exceptions) why the rows could not be parsed. Vertica recommends saving rejected data to a table.

### Multiple Rejected Data Files

Unless a load is very small (< 10MB), COPY creates more than one file to hold rejected rows. Several factors determine how many files COPY creates for rejected data. Here are some of the factors:

- Number of sources being loaded
- Total number of rejected rows
- Size of the source file (or files)
- Cooperative parsing and number of threads being used
- UDLs that support apportioned loads
- For your own COPY parser, the number of objects returned from `prepareUDSources()`

### Naming Conventions for Rejected Files

You can specify one or more rejected data files with the files you are loading. Use the REJECTED DATA parameter to specify a file location and name, and separate consecutive
rejected data file names with a comma (,). Do not use the ON ANY NODE option because it is applicable only to load files.

If you specify one or more files, and COPY requires multiple files for rejected data, COPY uses the rejected data file names you supply as a prefix, and appends a numeric suffix to each rejected data file. For example, if you specify the name myRejects for the REJECTED_DATA parameter, and the file you are loading is large enough (> 10MB), several files such as the following will exist:

- my_rejects-1
- my_rejects-2
- my_rejects-3

COPY uses cooperative parsing by default, having the nodes parse a specific part of the file contents. Depending on the file or portion size, each thread generates at least one rejected data file per source file or portion, and returns load results to the initiator node. The file suffix is a thread index when COPY uses multiple threads (.1, .2, .3, and so on).

The maximum number of rejected data files cannot be greater than the number of sources being loaded, per thread to parse any portion. The resource pool determines the maximum number of threads. For cooperative parse, use all available threads.

If you use COPY with a UDL that supports apportioned load, the file suffix is an offset value. UDL's that support apportioned loading render cooperative parsing unnecessary. For apportioned loads, COPY creates at least one rejected file per data portion, and more files depending on the size of the load and number of rejected rows.

For all data loads except COPY LOCAL, COPY behaves as follows:

<table>
<thead>
<tr>
<th>No rejected data file specified...</th>
<th>Rejected data file specified...</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a single data file (<em>pathToData</em> or STDIN), COPY stores one or more rejected data files in the default location.</td>
<td>For one data file, COPY interprets the rejected data path as a file, and stores all rejected data at the location. If more than one files is required from parallel processing, COPY appends a numeric suffix. If the path is not a file, COPY returns an error.</td>
</tr>
<tr>
<td>For multiple source files, COPY stores all rejected data in separate files in the default directory, using the source file as a file name prefix, as noted.</td>
<td>For multiple source files, COPY interprets the rejected path as a directory. COPY stores all information in separate files, one for each source. If path is not a directory, COPY returns an error.</td>
</tr>
<tr>
<td>No rejected data file specified...</td>
<td>Rejected data file specified...</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>COPY accepts only one path per node. For example, if you specify the rejected data path as my_rejected_data, COPY creates a directory of that name on each node. If you provide more than one rejected data path, COPY returns an error.</td>
<td></td>
</tr>
</tbody>
</table>

Rejected data files are returned to the initiator node.  

Rejected data files are not shipped to the initiator node.

**Maximum Length of File Names**

Loading multiple input files in one statement requires specifying full path names for each file. Keep in mind that long input file names, combined with rejected data file names, can exceed the operating system's maximum length (typically 255 characters). To work around file names that exceed the maximum length, use a path for the rejected data file that differs from the default path—for example, \tmp\<shorter-file-name>.

**Saving Rejected Data To a Table**

Use the REJECTED DATA parameter with the AS TABLE clause to specify a table in which to save rejected data. Saving rejected data to a file is mutually exclusive with using the AS TABLE clause.

When you use the AS TABLE clause, Vertica creates a new table if one does not exist, or appends to an existing table. If no parsing rejections occur during a load, the table exists but is empty. The next time you load data, Vertica inserts any rejected rows to the existing table.

The load rejection tables are a special type of table with the following capabilities and limitations:

- Support SELECT statements
- Can use DROP TABLE
- Cannot be created outside of a COPY statement
- Do not support DML and DDL activities
- Are not K-safe
To make the data in a rejected table K-safe, you can do one of the following:

- Write a CREATE TABLE..AS statement, such as this example:

  ```sql
  => CREATE TABLE new_table AS SELECT * FROM rejected_table;
  ```

- Create a table to store rejected records, and run INSERT..SELECT operations into the new table

**Using COPY NO COMMIT**

If the COPY statement includes options NO COMMIT and REJECTED DATA AS TABLE, and the `reject-table` does not already exist, Vertica Analytic Database saves the rejected data table as a LOCAL TEMP table and returns a message that a LOCAL TEMP table is being created.

Rejected-data tables are useful for Extract-Load-Transform workflows, where you will likely use temporary tables more frequently. The rejected-data tables let you quickly load data and identify which records failed to load. If you load data into a temporary table that you created using the ON COMMIT DELETE clause, the COPY operation will not commit.

**Location of Rejected Data Table Records**

When you save rejected records to a table, using the REJECTED DATA AS TABLE `table_name` option, the data for the table is saved in a database data subdirectory, `RejectionTableData`. For example, for a VMart database, table data files reside here:

```
/home/dbadmin/VMart/v_mlart_node0001_data/RejectionTableData
```

Rejected data tables include both rejected data and the reason for the rejection (exceptions), along with other data columns, described next. Vertica suggests that you periodically drop any rejected data tables that you no longer require.

**Querying a Rejected Data Table**

When you specify a rejected data table when loading data with COPY, you can query that table for information about rejected data after the load operation is complete. For example:

1. Create the `loader` table:

   ```sql
   => CREATE TABLE loader(a INT)
   CREATE TABLE
   ```
2. Use COPY to load values, saving rejected data to a table, loader_rejects:

```sql
COPY loader FROM STDIN REJECTED DATA AS TABLE loader_rejects;
```

Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.

```
>> 1
>> 2
>> 3
>> a
>> \.
```

3. Query the loader table after loading data:

```sql
SELECT * FROM loader;
```

```text
x
---
1
2
3
(3 rows)
```

4. Query the loader_rejects table to see its column rows:

```sql
SELECT * FROM loader_rejects;
```

```
<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmart_node0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>STDIN</td>
</tr>
<tr>
<td>session_id</td>
<td>v_vmart_node0001.example.-24016:0x3439</td>
</tr>
<tr>
<td>transaction_id</td>
<td>45035996274080923</td>
</tr>
<tr>
<td>statement_id</td>
<td>1</td>
</tr>
<tr>
<td>batch_number</td>
<td>0</td>
</tr>
<tr>
<td>row_number</td>
<td>4</td>
</tr>
<tr>
<td>rejected_data</td>
<td>a</td>
</tr>
<tr>
<td>rejected_data_orig_length</td>
<td>1</td>
</tr>
<tr>
<td>rejected_reason</td>
<td>Invalid integer format 'a' for column 1 (x)</td>
</tr>
</tbody>
</table>
```

The rejected data table has the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_name</td>
<td>VARCHAR</td>
<td>The name of the Vertica node on which the input load file was located.</td>
</tr>
<tr>
<td>file_name</td>
<td>VARCHAR</td>
<td>The name of the file being loaded, which applies if you loaded a file (as opposed to using STDIN).</td>
</tr>
<tr>
<td>session_id</td>
<td>VARCHAR</td>
<td>The session ID number in</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>transaction_id</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>statement_id</td>
<td>INTEGER</td>
<td>The unique identification number of the statement within the transaction that included the rejected data.</td>
</tr>
<tr>
<td>batch_number</td>
<td>INTEGER</td>
<td>INTERNAL USE. Represents which batch (chunk) the data comes from.</td>
</tr>
<tr>
<td>row_number</td>
<td>INTEGER</td>
<td>The rejected row number from the input file.</td>
</tr>
<tr>
<td>rejected_data</td>
<td>LONG VARCHAR</td>
<td>The data that was not loaded.</td>
</tr>
<tr>
<td>rejected_data_orig_length</td>
<td>INTEGER</td>
<td>The length of the rejected data.</td>
</tr>
</tbody>
</table>

Tip: You can use the session_id, transaction_id, and statement_id columns to create joins with many system tables. For example, if you join against the QUERY_REQUESTS table using those three columns, the QUERY_REQUESTS.REQUEST column contains the actual COPY statement (as a string) used to load this data.
### Exporting the Rejected Records Table

You can export the contents of the column `rejected_reason` to a file to capture only the data rejected during the first COPY statement. Then, correct the data in the file, save it, and load the updated file.

**To export rejected records:**

1. **Create a sample table:**

   ```sql
   => CREATE TABLE t (i int);
   CREATE TABLE
   ```

2. **Copy data directly into the table, using a table to store rejected data:**

   ```sql
   => COPY t FROM STDIN REJECTED DATA AS TABLE t_rejects;
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> 1
   >> 2
   >> 3
   >> 4
   >> a
   >> b
   >> c
   >> 
   ```

3. **Show only tuples and set the output format:**

   ```sql
   => \t
   Showing only tuples.
   => \a
   Output format is unaligned.
   ```

4. **Output to a file:**
5. Use the `cat` command on the saved file:

```sql
=> \! cat rejected.txt
a
b
c
```

After a file exists, you can fix load errors and use the corrected file as load input to the `COPY` statement.

## Saving Load Exceptions (EXCEPTIONS)

COPY exceptions consist of informational messages describing why a row of data could not be parsed. The optional `EXCEPTIONS` parameter lets you specify a file to which COPY writes exceptions. If you do not use this parameter, COPY saves exception files to the following default location:

`catalog_dir/CopyErrorLogs/tablename-filename-of-source-copy-from-exceptions`

<table>
<thead>
<tr>
<th><code>catalog_dir</code></th>
<th>The database catalog files directory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablename-filename-of-source</code></td>
<td>The names of the table and data file</td>
</tr>
<tr>
<td><code>-copy-from-exceptions</code></td>
<td>The file suffix appended to the table and source file name</td>
</tr>
</tbody>
</table>

**Note:** Using `REJECTED DATA AS TABLE r_table` is mutually exclusive with using the `EXCEPTIONS filename` parameter. The rejected data table includes a column with the exceptions messages. COPY does not permit both parameters. Trying to do so results in this error:

```
ERROR 0: Cannot specify both an exceptions file and a rejected table in the same statement
```

The optional `EXCEPTIONS` parameter lets you specify a file of your choice to which COPY writes load exceptions. The `EXCEPTIONS` file indicates the input line number and the reason for each data record exception in this format:

```
COPY: Input record number in <pathofinputfile> has been rejected (reason). Please see pathotrejectfile, record recordnum for the rejected record.
```

If copying from `STDIN`, the `filename-of-source` is `STDIN`. 
Note: You can use specific rejected data and exceptions files with one or more of the files you are loading. Separate consecutive rejected data and exception file names with a comma (,) in the COPY statement.

You must specify a filename in the path to load multiple input files. Keep in mind that long table names combined with long data file names can exceed the operating system's maximum length (typically 255 characters). To work around file names exceeding the maximum length, use a path for the exceptions file that differs from the default path; for example, \tmp\<shorter-file-name>.

For all data loads (except for COPY LOCAL), COPY behaves as follows:

<table>
<thead>
<tr>
<th>No Exceptions file specified...</th>
<th>Exceptions file specified...</th>
</tr>
</thead>
<tbody>
<tr>
<td>For one data source file (pathToData or STDIN), all information stored as one file in the default directory.</td>
<td>For one data file, the path is treated as a file, and COPY stores all information in this file. If the path is not a file, COPY returns an error.</td>
</tr>
<tr>
<td>For multiple data files, all information stored as separate files, one for each data file in default directory.</td>
<td>For multiple data files, the path is treated as a directory and COPY stores all information in separate files, one for each data file. If path is not a directory, COPY returns an error.</td>
</tr>
<tr>
<td></td>
<td>Exceptions files are not stored on the initiator node.</td>
</tr>
<tr>
<td></td>
<td>You can specify only one path per node. If you specify more than one path per node, COPY returns an error.</td>
</tr>
</tbody>
</table>

COPY Rejected Data and Exception Files

When executing a COPY statement, and parallel processing is ON (the default setting), COPY creates separate threads to process load files. Typically, the number of threads depends on the number of node cores in the system. Each node processes part of the load data. If the load succeeds overall, any parser rejections that occur during load processing are written to that node's specific rejected data and exceptions files. If the load fails, the rejected data file contents can be incomplete, or empty. If you do not specify a file name explicitly, COPY uses a default name and location for rejected data files. See the next topic for specifying your own rejected data and exception files.

Both rejected data and exceptions files are saved and stored on a per-node basis. This example uses multiple files as COPY inputs. Since the statement does not include either the REJECTED
DATA or EXCEPTIONS parameters, rejected data and exceptions files are written to the default location, the database catalog subdirectory, CopyErrorLogs, on each node:

```sql
\set dir `pwd`/data/ \set remote_dir /vertica/test_dev/tmp_ms/
\set file1 ''':dir'C1_large_t1.dat'''
\set file2 ''':dir'C2_large_t1.dat'''
\set file3 ''':remote_dir'C3_large_t1.dat'''
\set file4 ''':remote_dir'C4_large_t1.dat'''
=>COPY large_t1 FROM :file1 ON site01,:file2 ON site01,
   :file3 ON site02,
   :file4 ON site02
   DELIMITER '|';
```

### Specifying Rejected Data and Exceptions Files

The optional COPY REJECTED DATA and EXCEPTIONS parameters 'path' element lets you specify a non-default path in which to store the files.

If *path* resolves to a storage location, and the user invoking COPY is not a superuser, these are the required permissions:

- The storage location must have been created (or altered) with the USER option (see CREATE LOCATION and ALTER_LOCATION_USE)

- The user must already have been granted READ access to the storage location where the file(s) exist, as described in GRANT (Storage Location)

Both parameters also have an optional ON nodename clause that uses the specified path:

```
...[ EXCEPTIONS 'path' [ ON nodename ] [, ... ] ]...[ REJECTED DATA 'path' [ ON nodename ] [, ... ] ]
```

While 'path' specifies the location of the rejected data and exceptions files (with their corresponding parameters), the optional ON nodename clause moves any existing rejected data and exception files on the node to the specified path on the same node.

### Saving Rejected Data and Exceptions Files to a Single Server

The COPY statement does not have a facility to merge exception and rejected data files after COPY processing is complete. To see the contents of exception and rejected data files requires accessing each node's specific files.

**Note:** To save all exceptions and rejected data files on a network host, be sure to give each node's files unique names, so that different cluster nodes do not overwrite other nodes' files. For instance, if you set up a server with two directories (/vertica/exceptions
and /vertica/rejections), specify file names for each Vertica cluster node to identify each node, such as node01_exceptions.txt and node02_exceptions.txt. This way, each cluster node's files are easily distinguishable in the exceptions and rejections directories.

Using VSQL Variables for Rejected Data and Exceptions Files

This example uses vsql variables to specify the path and file names to use with the exceptions and rejected data parameters (except_s1 and reject_s1). The COPY statement specifies a single input file (large_tbl) on the initiator node:

```sql
\set dir `pwd'/data/ \set file1 '''':dir'C1_large_tbl.dat'''
\set except_s1 '''':dir'exceptions'''
\set reject_s1 '''':dir'rejections'''
COPY large_tbl FROM :file1 ON site01 DELIMITER '|'
REJECTED DATA :reject_s1 ON site01
EXCEPTIONS :except_s1 ON site01;
```

This example uses variables to specify exception and rejected date files (except_s2 and reject_s2) on a remote node. The COPY statement consists of a single input file on a remote node (site02):

```sql
\set remote_dir /vertica/test_dev/tmp_ms/\set except_s2 '''':remote_dir'exceptions'''
\set reject_s2 '''':remote_dir'rejections'''
COPY large_tbl FROM :file1 ON site02 DELIMITER '|'
REJECTED DATA :reject_s2 ON site02
EXCEPTIONS :except_s2 ON site02;
```

This example uses variables to specify that the exception and rejected data files are on a remote node (indicated by :remote_dir). The inputs to the COPY statement consist of multiple data files on two nodes (site01 and site02). The exceptions and rejected data options use the ON  nodename clause with the variables to indicate where the files reside (site01 and site02):

```sql
\set dir `pwd'/data/ \set remote_dir /vertica/test_dev/tmp_ms/
\set except_s1 '''':dir'''
\set reject_s1 '''':dir'''
\set except_s2 '''':remote_dir'''
\set reject_s2 '''':remote_dir'''
COPY large_tbl FROM :file1 ON site01,
    :file2 ON site01,
    :file3 ON site02,
    :file4 ON site02
DELIMITER '|'
REJECTED DATA :reject_s1 ON site01, :reject_s2 ON site02
EXCEPTIONS :except_s1 ON site01, :except_s2 ON site02;
```
COPY LOCAL Rejection and Exception Files

Invoking COPY LOCAL (or COPY LOCAL FROM STDIN) does not automatically create rejected data and exceptions files. This behavior differs from using COPY, which saves both files automatically, regardless of whether you use the optional REJECTED DATA and EXCEPTIONS parameters to specify either file explicitly.

Use the REJECTED DATA and EXCEPTIONS parameters with COPY LOCAL and COPY LOCAL FROM STDIN to save the corresponding output files on the client. If you do not use these options, rejected data parsing events (and the exceptions that describe them) are not retained, even if they occur.

You can load multiple input files using COPY LOCAL (or COPY LOCAL FROM STDIN). If you also use the REJECTED DATA and EXCEPTIONS options, the statement writes rejected rows and exceptions and to separate files. The respective files contain all rejected rows and corresponding exceptions, respectively, regardless of how many input files were loaded.

If COPY LOCAL does not reject any rows, it does not create either file.

Note: Because COPY LOCAL (and COPY LOCAL FROM STDIN) must write any rejected rows and exceptions to the client, you cannot use the [ON nodename ] clause with either the rejected data or exceptions options.

Specifying Rejected Data and Exceptions Files

To save any rejected data and their exceptions to files:

1. In the COPY LOCAL (and COPY LOCAL FROM STDIN) statement, use the REJECTED DATA 'path' and the EXCEPTIONS 'path' parameters, respectively.

2. Specify two different file names for the two options. You cannot use one file for both the REJECTED DATA and the EXCEPTIONS.

3. When you invoke COPY LOCAL or COPY LOCAL FROM STDIN, the files you specify need not pre-exist. If they do, COPY LOCAL must be able to overwrite them.

You can specify the path and file names with vsqI variables:

```
\set rejected ../../../except_reject/copyLocal.rejected
\set exceptions ../../../except_reject/copyLocal.exceptions
```
**Note**: Using `COPY LOCAL` does not support storing rejected data in a table, as you can when using the `COPY` statement.

When you use the `COPY LOCAL` or `COPY LOCAL FROM STDIN` statement, specify the variable names for the files with their corresponding parameters:

```plaintext
=> COPY large_tbl FROM LOCAL rejected data :rejected exceptions :exceptions;
=> COPY large_tbl FROM LOCAL STDIN rejected data :rejected exceptions :exceptions;
```

## Referential Integrity Load Violation

Vertica checks for constraint violations during query execution. It checks for violations during load operations only if you enable primary or unique key enforcement. In this case, Vertica checks for referential integrity constraint violations when loading data.

If Vertica detects a referential integrity constraint violation, the load rolls back and Vertica returns an error.

### See Also

- Detecting Constraint Violations with `ANALYZE_CONSTRAINTS`
- `COPY`
- `ANALYZE_CONSTRAINTS`
Trickle Loading Data

After an initial bulk data load is complete, use COPY...TRICKLE to load data incrementally with new and changed data.

By default, your database uses transaction isolation level READ COMMITTED. Using this isolation level has these benefits:

- Users can see the most recently committed data without holding any locks.
- New data can be loaded while concurrent queries are running.

See Change Transaction Isolation Levels.

Using INSERT, UPDATE, and DELETE

DML statements INSERT, UPDATE, and DELETE perform the same functions that they do in any ACID compliant database. INSERT is typically used to load individual rows into physical memory or load a table using an INSERT AS SELECT clause. UPDATE and DELETE are used to modify the data.

You can intermix INSERT, UPDATE, and DELETE. Vertica follows the SQL-92 transaction model. In other words, you do not have to explicitly start a transaction but you must use a COMMIT or ROLLBACK command (or COPY) to end a transaction. If you cancel a DML statement, Vertica rolls back the statement.

Vertica differs from traditional databases in two ways:

- DELETE does not actually delete data from disk storage; it marks rows as deleted so they are available for historical queries.
- UPDATE writes two rows: one with new data and one marked for deletion.

Unless you specify otherwise, INSERT, UPDATE and DELETE statements write data to the WOS. On overflow, the statements write data to the ROS.

Tip: For large INSERT or UPDATE operations, use the DIRECT hint. This hint forces Vertica to load all data directly to ROS. Loading large number of rows as single row inserts is not recommended for performance reasons. Instead, use COPY.
Use of WOS for Trickle Loads

During trickle loading, the WOS (node memory) permits the Vertica to efficiently batch small data loads into larger ones for I/O purposes. Loading to WOS is significantly faster than loading to disk, but it has size constraints. To help balance speed, memory size, and data flow, trickle loading defers the work of sorting, encoding, and writing to disk. The Tuple Mover's moveout process performs those tasks in the background when a table or DML operation uses trickle loading as its load method.

WOS capacity is significantly less than ROS. When incoming data exceeds WOS capacity, Vertica spills small loads directly to disk. While no data is lost when the WOS gets full and spills to ROS, it can result in wasted I/O bandwidth. For optimal performance, follow the guidelines described in Managing the Tuple Mover to avoid WOS overflow.

If the WOS becomes full, before the tuple mover can move data to ROS, the entire load fails and is rolled back.
Importing and Exporting Data Across Databases

Vertica can easily import data from and export data to other Vertica databases. Importing and exporting data is useful for common tasks such as moving data back and forth between a development or test database and a production database, or between databases that have different purposes but need to share data on a regular basis.

Moving Data Directly Between Databases

Use the following statements move data to and from another Vertica database:

- COPY FROM VERTICA
- EXPORT TO VERTICA

To execute either of these statements requires first creating a connection to the other Vertica database.

Creating SQL Scripts to Export Data

Three functions return a SQL script you can use to export database objects to recreate elsewhere:

- EXPORT_CATALOG
- EXPORT_OBJECTS
- EXPORT_TABLES

While copying and exporting data is similar to Backing Up and Restoring the Database, you should use them for different purposes, outlined below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Backup and Restore</th>
<th>COPY and EXPORT Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back up or restore an entire database, or</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
The following sections explain how you import and export data between Vertica databases.

When importing from or exporting to a Vertica database, you can connect only to a database that uses trusted- (username-only) or password-based authentication, as described in Security and Authentication. SSL authentication is not supported.

### Other Exports

This section is about exporting data to another Vertica database. For information about exporting data to Parquet files in HDFS, see Exporting Data in Integrating with Apache Hadoop.

### Exporting Data to Another Vertica Database

You can export a table, specific columns in a table, or the results of a `SELECT` statement to another Vertica database. The table in the target database receiving the exported data must already exist, and have columns that match (or can be coerced to) the data types of the columns you are exporting.

You can import data from an earlier Vertica release, if the earlier release is a version of the last major release before the target database release.

Exported data is always written in AUTO mode. For details, see Choosing a Load Method.
Export Process

Exporting is a three-step process:

1. **CONNECT** connects to the target database.

   Caution: The export operation fails if either side of the connection is a single-node cluster installed to localhost, or you do not specify a host name or IP address.

2. **EXPORT TO VERTICA** exports the data. You can export only one table at a time. Use multiple EXPORT statements to export multiple tables or the results of multiple SELECT statements. All statements use the same connection to the target database.

3. **DISCONNECT** disconnects from the target database when the export operation is complete.

Exporting Identity Columns

You can export tables (or columns) that contain identity and auto-increment values, but the sequence values are not incremented automatically at the target table. You must use `ALTER SEQUENCE` to make updates.

Export identity (and auto-increment) columns as follows:

- If both source and destination tables have an identity column and configuration parameter `CopyFromVerticaWithIdentity` is set to true (1), you do not need to list them.

- If source table has an identity column, but target table does not, you must explicitly list the source and target columns.

  Caution: Failure to list which identity columns to export can cause an error, because the identity column will be interpreted as missing in the destination table.

By default, **EXPORT TO VERTICA** exports all identity columns. To disable this behavior globally, set the `CopyFromVerticaWithIdentity` configuration parameter.
Exporting GEOMETRY and GEOGRAPHY Data Types

You can export columns containing Vertica geospatial data types. Vertica supports the following export methods for these data types:

<table>
<thead>
<tr>
<th>Export/Copy Support</th>
<th>Target version: 8.0 SP1 and later</th>
<th>Target version: Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source version: 8.0 SP1 and later</td>
<td>Export/Copy</td>
<td>Export</td>
</tr>
<tr>
<td>Source version: Previous</td>
<td>Copy</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Examples of Exporting Data

The following example demonstrates using the three-step process listed above to export data.

First, open the connection to the other database, then perform a simple export of an entire table to an identical table in the target database.

```
=> CONNECT TO VERTICA testdb USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> EXPORT TO VERTICA testdb.customer_dimension FROM customer_dimension;
Rows Exported
-------------
23416
(1 row)
```

The following statement demonstrates exporting a portion of a table using a simple SELECT statement.

```
=> EXPORT TO VERTICA testdb.ma_customers AS SELECT customer_key, customer_name, annual_income FROM customer_dimension WHERE customer_state = 'MA';
Rows Exported
-------------
3429
(1 row)
```

This statement exports several columns from one table to several different columns in the target database table using column lists. Remember that when supplying both a source and destination column list, the number of columns must match.

```
=> EXPORT TO VERTICA testdb.people (name, gender, age) FROM customer_dimension (customer_name, customer_gender, customer_age);
```
You can also use EXPORT TO VERTICA with a SELECT AT EPOCH LATEST expression to include data from the latest committed DML transaction.

Disconnect from the database when the export is complete:

=> DISCONNECT testdb;
DISCONNECT

Note: Closing your session also closes the database connection. However, it is a good practice to explicitly close the connection to the other database, both to free up resources and to prevent issues with other SQL scripts you may run in your session. Always closing the connection prevents potential errors if you run a script in the same session that attempts to open a connection to the same database, since each session can only have one connection to a particular database at a time.

## Copying Data from Another Vertica Database

You can import a table or specific columns in a table from another Vertica database. The table receiving the copied data must already exist, and have columns that match (or can be coerced into) the data types of the columns you are copying from the other database. You can import data from an earlier Vertica release, if the earlier release is a version of the last major release before the target database release.

### Import Process

Importing is a three-step process:

1. **CONNECT** connects to the database that contains the data to import.

   Note: The import operation fails if either side of the connection is a single-node cluster installed to localhost, or you do not specify a host name or IP address.
2. **COPY FROM VERTICA** imports table data to the target database. To import data from multiple tables, issue multiple **COPY FROM VERTICA** statements using the same connection to the source database.

3. **DISCONNECT** disconnects from the source database when the copy operation is complete.

### Importing Identity Columns

You can import identity (and auto-increment) columns as follows:

- If both source and destination tables have an identity column and configuration parameter `CopyFromVerticaWithIdentity` is set to true (1), you do not need to list them.

- If source table has an identity column, but target table does not, you must explicitly list the source and target columns.

**Caution:** Failure to list which identity columns to export can cause an error, because the identity column will be interpreted as missing in the destination table.

After importing the columns, the identity column values do not increment automatically. Use **ALTER SEQUENCE** to make updates.

The default behavior for this statement is to import Identity (and Auto-increment) columns by specifying them directly in the source table. To disable this behavior globally, set the `CopyFromVerticaWithIdentity` configuration parameter, described in **Configuration Parameters**.

### Examples

This example demonstrates connecting to another database, copying the contents of an entire table from the source database to an identically-defined table in the current database directly into ROS, and then closing the connection:

```sql
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY customer_dimension FROM VERTICA vmart.customer_dimension DIRECT;
Rows Loaded
-------------
  500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```
This example demonstrates copying several columns from a table in the source database into a table in the local database:

```sql
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY people (name, gender, age) FROM VERTICA
    vmart.customer_dimension (customer_name, customer_gender, customer_age);
Rows Loaded
-------------
500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

### Changing Node Export Addresses

You can change the export address for your Vertica cluster. You might need to do so to export data between clusters in different network subnets.

1. **Create a subnet for importing and exporting data between Vertica clusters.** The `CREATE SUBNET` statement identifies the public network IP addresses residing on the same subnet.

   ```sql
   => CREATE SUBNET kv_subnet with '10.10.10.0';
   ```

2. **Alter the database to specify the subnet name of a public network for import/export.**

   ```sql
   => ALTER DATABASE VMartDB EXPORT ON kv_subnet;
   ```

3. **Create network interfaces for importing and exporting data from individual nodes to other Vertica clusters.** The `CREATE NETWORK INTERFACE` statement identifies the public network IP addresses residing on multiple subnets.

   ```sql
   => CREATE NETWORK INTERFACE kv_node1 on v_VMartDB_node0001 with '10.10.10.1';
   => CREATE NETWORK INTERFACE kv_node2 on v_VMartDB_node0002 with '10.10.10.2';
   => CREATE NETWORK INTERFACE kv_node3 on v_VMartDB_node0003 with '10.10.10.3';
   => CREATE NETWORK INTERFACE kv_node4 on v_VMartDB_node0004 with '10.10.10.4';
   ```

   For users on Amazon Web Services (AWS) or using Network Address Translation (NAT), refer to [Vertica on Amazon Web Services](https://www.vertica.com/docs/9.0.x/EN-G00306203.html).

4. **Alter the node settings to change the export address.** When used with the `EXPORT ON` clause, the `ALTER NODE` specifies the network interface of the public network on individual nodes.
nodes for importing and exporting data.

```sql
=> ALTER NODE v_VMartDB_node0001 export on kv_node1;
=> ALTER NODE v_VMartDB_node0002 export on kv_node2;
=> ALTER NODE v_VMartDB_node0003 export on kv_node3;
=> ALTER NODE v_VMartDB_node0004 export on kv_node4;
```

5. Verify if the node address and the export address are different on different network subnets of the Vertica cluster.

```sql
=> SELECT node_name, node_address, export_address FROM nodes;
node_name | node_address | export_address
---------- |------------- |--------------
v_VMartDB_node0001 | 192.168.100.101 | 10.10.10.1
v_VMartDB_node0002 | 192.168.100.102 | 10.10.10.2
v_VMartDB_node0003 | 192.168.100.103 | 10.10.10.3
v_VMartDB_node0004 | 192.168.100.104 | 10.10.10.4
```

Creating a network interface and altering the node settings to change the export address takes precedence over creating a subnet and altering the database for import/export.
Using Public and Private IP Networks

In many configurations, Vertica cluster hosts use two network IP addresses as follows:

- A private address for communication between the cluster hosts.
- A public IP address for communication with client connections.

By default, importing from and exporting to another Vertica database uses the private network.

**Note:** Ensure port 5433 or the port the Vertica database is using is not blocked.

To use the public network address for copy and export activities, as well as moving large amounts of data, configure the system to use the public network to support exporting to or importing from another Vertica cluster:

- **Identify the Public Network to Vertica**
- **Identify the Database or Nodes Used for Import/Export**

In certain instances, both public and private addresses exceed the demand capacity of a single Local Area Network (LAN). If you encounter this type of scenario, then configure your Vertica cluster to use two LANs: one for public network traffic and one for private network traffic.

Identify the Public Network to Vertica

To be able to import to or export from a public network, Vertica needs to be aware of the IP addresses of the nodes or clusters on the public network that will be used for import/export activities. Your public network might be configured in either of these ways:

- Public network IP addresses reside on the same subnet (create a subnet)
- Public network IP addresses are on multiple subnets (create a network interface)

To identify public network IP addresses residing on the same subnet:

- Use the **CREATE SUBNET** statement provide your subnet with a name and to identify the subnet routing prefix.

To identify public network IP addresses residing on multiple subnets:
Use the **CREATE NETWORK INTERFACE** statement to configure import/export from specific nodes in the Vertica cluster.

After you've identified the subnet or network interface to be used for import/export, you must **Identify the Database Or Nodes Used For Import/Export**.

### See Also

- **CREATE SUBNET**
- **ALTER SUBNET**
- **DROP SUBNET**
- **CREATE NETWORK INTERFACE**
- **ALTER NETWORK INTERFACE**
- **DROP NETWORK INTERFACE**

### Identify the Database or Nodes Used for Import/Export

After you identify the public network to Vertica, you can configure a database and its nodes to use it for import and export operations:

- Use **ALTER DATABASE** to specify a subnet on the public network for the database. After doing so, all nodes in the database automatically use the network interface on the subnet for import/export operations.

- On each database node, use **ALTER NODE** to specify a network interface of the public network.

### See Also

- **CREATE PROCEDURE**
- **CREATE NETWORK INTERFACE**
- **V_MONITOR.NETWORKINTERFACES**
Handling Node Failure During Copy/Export

When an export (EXPORT TO VERTICA) or import from Vertica (COPY FROM VERTICA) task is in progress, and a non-initiator node fails, Vertica does not complete the task automatically. A non-initiator node is any node that is not the source or target node in your export or import statement. To complete the task, you must run the statement again.

You address the problem of a non-initiator node failing during an import or export as follows:

**Note: Both Vertica databases must be running in a safe state.**

1. You export or import from one cluster to another using the EXPORT TO VERTICA or COPY FROM VERTICA statement.

   During the export or import, a non-initiating node on the target or source cluster fails. Vertica issues an error message that indicates possible node failure, one of the following:

   - **ERROR 4534**: Receive on v_tpcdb1_node0002: Message receipt from v_tpcdb2_node0005 failed
   - **WARNING 4539**: Received no response from v_tpcdb1_node0004 in abandon plan
   - **ERROR 3322**: [tpchdb2] Execution canceled by operator

2. Complete your import or export by running the statement again. The failed node does not need to be up for Vertica to successfully complete the export or import.
Using EXPORT Functions

Vertica provides several EXPORT_ functions that let you recreate a database, or specific schemas and tables, in a target database. For example, you can use the EXPORT_ functions to transfer some or all of the designs and objects you create in a development or test environment to a production database.

The EXPORT_ functions create SQL scripts that you can run to generate the exported database designs or objects. These functions serve different purposes to the export statements, COPY FROM VERTICA (pull data) and EXPORT TO VERTICA (push data). These statements transfer data directly from source to target database across a network connection between both. They are dynamic actions and do not generate SQL scripts.

The EXPORT_ functions appear in the following table. Depending on what you need to export, you can use one or more of the functions. EXPORT_CATALOG creates the most comprehensive SQL script, while EXPORT_TABLES and EXPORTOBJECTS are subsets of that function to narrow the export scope.

<table>
<thead>
<tr>
<th>Use this function...</th>
<th>To recreate...</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT_CATALOG</td>
<td>These catalog items:</td>
</tr>
<tr>
<td></td>
<td>- An existing schema design, tables, projections, constraints, and views</td>
</tr>
<tr>
<td></td>
<td>- The Database Designer-created schema design, tables, projections, constraints, and views</td>
</tr>
<tr>
<td></td>
<td>- A design on a different cluster.</td>
</tr>
<tr>
<td>EXPORT_TABLES</td>
<td>Non-virtual objects up to, and including, the schema of one or more tables.</td>
</tr>
<tr>
<td>EXPORT_OBJECTS</td>
<td>Catalog objects in order dependency for replication.</td>
</tr>
</tbody>
</table>

The designs and object definitions that the script creates depend on the EXPORT_ function scope you specify. The following sections give examples of the commands and output for each function and the scopes it supports.
Saving Scripts for Export Functions

All of the examples in this section were generated using the standard Vertica VMART database, with some additional test objects and tables. One output directory was created for all SQL scripts that the functions created:

```
/home/dbadmin/xtest
```

If you specify the destination argument as an empty string (''), the function writes the export results to STDOUT.

**Note:** A superuser can export all available database output to a file with the EXPORT_ functions. For a non-superuser, the EXPORT_ functions generate a script containing only the objects to which the user has access.

Exporting the Catalog

The function EXPORT_CATALOG generates a SQL script for copying a database design to another cluster. This script replicates the physical schema design of the source database. You call this function as follows:

```
EXPORT_CATALOG ([ 'destination' ], [ 'scope' ])
```

**Note:** This function and EXPORT_OBJECTS return equivalent output.

Setting Scope of Export

You can set the scope of the export operation to various levels:

<table>
<thead>
<tr>
<th>To export...</th>
<th>Use this scope...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schemas, tables, constraints, views, and projections</td>
<td>DESIGN (default)</td>
</tr>
<tr>
<td>All design objects and system objects created in Database Designer, such as design contexts and their tables</td>
<td>DESIGN ALL</td>
</tr>
<tr>
<td>Only tables, constraints, and projection</td>
<td>TABLES</td>
</tr>
</tbody>
</table>
Exporting All Catalog Objects

Use the DESIGN scope to export all design elements of a source database in order dependency. This scope exports all catalog objects in their OID (unique object ID) order, including schemas, tables, constraints, views, and all types of projections. This is the most comprehensive export scope, without the Database Designer elements, if they exist.

```sql
=> SELECT EXPORT_CATALOG(  
    '/home/dbadmin/xtest/sql_cat_design.sql',  
    'DESIGN' );  
---  
-- EXPORT_CATALOG  
---  
Catalog data exported successfully  
(1 row)
```

The SQL script includes the following types of statements, each needed to recreate a new database:

- CREATE SCHEMA
- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROJECTION (with ORDER BY and SEGMENTED BY)
- ALTER TABLE (to add constraints)
- PARTITION BY

Projection Considerations

If a projection to export was created with no ORDER BY clause, the SQL script reflects the default behavior for projections. Vertica implicitly creates projections using a sort order based on the SELECT columns in the projection definition. The EXPORT_CATALOG script reflects this behavior.

The EXPORT_CATALOG script is portable if all projections were generated using UNSEGMENTED ALL NODES or SEGMENTED ALL NODES.
Exporting Database Designer Schema and Designs

Use the DESIGN ALL scope to generate a script to recreate all design elements of a source database and the design and system objects that were created by the Database Designer:

```sql
=> SELECT EXPORT_CATALOG ('/home/dbadmin/xtest/sql_cat_design_all.sql', 'DESIGN_ALL');
EXPORT_CATALOG
------------------------------
Catalog data exported successfully
(1 row)
```

Exporting Table Objects

Use the TABLES scope to generate a script to recreate all schemas tables, constraints, and sequences:

```sql
=> SELECT EXPORT_CATALOG ('/home/dbadmin/xtest/sql_cat_tables.sql', 'TABLES');
EXPORT_CATALOG
------------------------------
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE

See Also

- EXPORT_CATALOG
- EXPORT_OBJECTS
- EXPORT_TABLES
Exporting Tables

Use the Vertica function `EXPORT_TABLES` to recreate one or more tables, and related non-virtual objects, on a different cluster. `EXPORT_TABLES` has the following syntax:

```sql
EXPORT_TABLES( [ 'destination' ] [, 'scope' ] )
```

Setting Scope of Export

You can set the scope of the export operation to various levels:

<table>
<thead>
<tr>
<th>To export...</th>
<th>Use this scope...</th>
</tr>
</thead>
<tbody>
<tr>
<td>All objects to which the user has access, including constraints.</td>
<td>An empty string (' ')</td>
</tr>
<tr>
<td>One or more named objects, such as tables or sequences in one or more schemas. You can optionally qualify the schema with a database prefix, <code>myvertica.myschema.newtable</code>.</td>
<td>A comma-delimited list of table objects. For example: 'myschema.newtable, yourschema.oldtable'</td>
</tr>
<tr>
<td>A named table object in the current search path. You can specify a schema, table, or sequence. If you specify a schema, <code>EXPORT_TABLES</code> exports all table objects in that schema to which the user has access.</td>
<td>The table object's name and, optionally, its path: 'VMart.myschema'</td>
</tr>
</tbody>
</table>

The SQL script includes only the non-virtual objects to which the current user has access.

**Note:** `EXPORT_TABLES` does not export views. If you specify a view name, Vertica silently ignores it and the view is omitted from the generated script. To export views, use `EXPORT_OBJECTS`.

Exporting All Table Objects

If you set the scope parameter to an empty string (' '), Vertica exports all tables and their related objects:
The SQL script includes the following types of statements, depending on what is required to recreate the tables and any related objects (such as sequences):

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE
- PARTITION BY

Exporting a List of Tables

Use EXPORT_TABLE with a comma-separated list of objects, including tables, views, or schemas:

```sql
SELECT EXPORT_TABLES('HOME/dbadmin/xtest/sql_tables_empty.sql', '');
```

Catalog data exported successfully
(1 row)

The SQL script can include the following types of statements, depending on what is required to recreate the exported objects:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE
Exporting a Single Table Object

Use EXPORT_TABLES to export one or more database table objects.

This example exports a named sequence, my_seq, qualifying the sequence with the schema name (public):

```sql
=> SELECT EXPORT_TABLES('home/dbadmin/xtest/export_one_sequence.sql', 'public.my_seq');
EXPORT_TABLES
-------------------------------------
Catalog data exported successfully
(1 row)
```

Following are the contents of the export_one_sequence.sql output file using a more command:

```
$ more export_one_sequence.sql
CREATE SEQUENCE public.my_seq ;
```

See Also

Exporting Objects

Exporting Objects

The Vertica function EXPORT_OBJECTS generates a SQL script that you can use to recreate non-virtual catalog objects on a different cluster, as follows:

```sql
EXPORT_OBJECTS( [ 'destination' ] [, 'scope' ] [, 'ksafe' ] )
```

**Note:** This function and EXPORT_CATALOG return equivalent output.

Setting Scope of Export

You can set the scope of the export operation to various levels:

<table>
<thead>
<tr>
<th>To export...</th>
<th>Use this scope...</th>
</tr>
</thead>
<tbody>
<tr>
<td>All non-virtual objects to which the user has access, including constraints.</td>
<td>An empty string ('')</td>
</tr>
</tbody>
</table>
To export... | Use this scope...
---|---
One or more named objects, such as tables or views in one or more schemas. You can optionally qualify the schema with a database prefix, myvertica.myschema.newtable. | A comma-delimited list of items. For example: 'myschema.newtable, yourschema.oldtable'
A named database object in the current search path. You can specify a schema, table, or view. If the object is a schema, the script includes objects to which the user has access. | The table object's name and, optionally, its path: 'VMart.myschema'

The SQL script includes only the non-virtual objects to which the current user has access.

EXPORT_OBJECTS always tries to recreate projection statements with their KSAFE clauses, if any; otherwise, with their OFFSET clauses.

Function Syntax

```sql
EXPORT_OBJECTS( [ 'destination' ] , [ 'scope' ] , [ 'ksafe' ] )
```

Exporting All Objects

If you set the scope parameter to an empty string (''), Vertica exports all non-virtual objects from the source database in order dependency. Running the generated SQL script on another cluster creates all referenced objects and their dependent objects.

By default, the function's KSAFE argument is set to true. In this case, the generated script calls MARKDESIGN_KSAFE, which propagates the K-safety setting of the original database.

```sql
=> SELECT EXPORT_OBJECTS(
   '/home/dbadmin/xtest/sql_objects_all.sql',
   '',
   'true');
EXPORT_OBJECTS
-------------------------------
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE
Here is a snippet that includes the start and end of the output SQL file, including the MARK_DESIGN_KSAFE statement:

```sql
CREATE SCHEMA store;
CREATE SCHEMA online_sales;
CREATE SEQUENCE public.my_seq;
CREATE TABLE public.customer_dimension
(
    customer_key int NOT NULL,
    customer_type varchar(16),
    customer_name varchar(256),
    customer_gender varchar(8),
    title varchar(8),
    household_id int,
...
);

SELECT MARK_DESIGN_KSAFE(1);
```

**Exporting a List of Objects**

Use a comma-separated list of objects as the function scope. The list can include one or more tables, sequences, and views in the same, or different schemas, depending on how you qualify the object name. For instance, specify a table from one schema, and a view from another (schema2.view1).

The SQL script includes the following types of statements, depending on what objects you include in the list:

- **CREATE SCHEMA**
- **CREATE TABLE**
- **ALTER TABLE (to add constraints)**
CREATE VIEW
CREATE SEQUENCE

If you specify a view without its dependencies, the function displays a WARNING. The SQL script includes a CREATE statement for the dependent object, but will be unable to create it without the necessary relations:

```sql
=> SELECT EXPORT_OBJECTS('nameObjectsList',
   'test2, tt, my_seq, v2 ');
WARNING 0: View public.v2 depends on other relations
EXPORT_OBJECTS
-----------------------------
Catalog data exported successfully
(1 row)
```

This example explicitly sets the `ksafe` argument explicitly to `true`.

```sql
=> SELECT EXPORT_OBJECTS('
   /home/dbadmin/xtest/sql_objects_table_view_KSAFE.sql',
   'v1, test7',
   'true');
EXPORT_OBJECTS
-----------------------------
Catalog data exported successfully
(1 row)
```

Here are the contents of the output file of the example, showing the sample table `test7` and the `v1` view:

```sql
CREATE TABLE public.test7
(
   a int,
   c int NOT NULL DEFAULT 4,
   bb int
);
CREATE VIEW public.v1 AS
SELECT tt.a
FROM public.tt;
SELECT MARK_DESIGN_KSAFE(1);
```

Exporting a Single Object

Specify a single database object as the function scope. The object can be a schema, table, sequence, or view. The function exports all non-virtual objects associated with the one you specify.
=> SELECT EXPORT_OBJECTS(
  '/home/dbadmin/xtest/sql_objects_viewobject_KSAFE.sql',
  'v1',
  'true');

EXPORT_OBJECTS
-----------------------------
Catalog data exported successfully
(1 row)

The output file contains the v1 view:

CREATE VIEW public.v1 AS
  SELECT tt.a
  FROM public.tt;

SELECT MARK DESIGN_KSAFE(1);

See Also

Exporting Tables
**Managing Storage Locations**

Vertica storage locations are the paths to file destinations you designate to store data and temporary files. Every node in the cluster requires at least one area in which to store data and another separate area in which to store database catalog files. You set up these locations as part of installation and setup. (See Prepare Disk Storage Locations in Installing Vertica for disk space requirements.)

**Important:** While no technical issue prevents you from using CREATE LOCATION to add one or more Network File System (NFS) storage locations, Vertica does not support NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify ALL NODES SHARED.

**How Vertica Uses Storage Locations**

Every time you add data to the database or perform a DML operation, the new data is held in memory (WOS) and moved to storage locations on disk (ROS) at regular intervals. Depending on the configuration of your database, many ROS containers are likely to exist.

You can label the storage locations you create to reference them in object storage policies. If an object to store has no associated storage policy, Vertica uses the available storage location and default storage algorithms to store data. If the object has a storage policy, Vertica stores the data at the object’s designated storage location. See Creating Storage Policies. If you no longer need a storage location, you can retire or drop it, as described in Retiring Storage Locations and Dropping Storage Locations.

**Comparing Local and Shared Storage Locations**

You can define two types of storage locations:

- **Local**—Stores data in a unique location for each node. Local is the default for a storage location. You do not have to specify it explicitly. This location is in a directory in a file system that the node can access, and is often in the node’s own file system. You can create a local...
storage location for a single node or for all nodes in the cluster. Cluster-wide storage locations are the most common type of storage. Vertica defaults to using a local cluster-wide storage location for storing all data. If you want it to store data differently, you must create additional storage locations.

- **Shared**—Stores data on a single file system to which all nodes in the cluster have access. This shared file system is often hosted outside of the cluster, such as on a distributed file system like HDFS. When you create a shared storage location, each node in the Vertica cluster creates its own subdirectory in the location. The separate directories prevent nodes from overwriting each other's data. Currently, Vertica supports only HDFS shared storage locations. You cannot use NFS as a Vertica shared storage location except when using MapR mount points. See Vertica Storage Location for HDFS in Integrating with Apache Hadoop for more information.

For databases running in Eon Mode Beta, the STORAGE_LOCATIONS system table shows a third type of location, communal.

**Viewing Storage Locations and Policies**

You can monitor information about available storage location labels and your current storage policies.

**Viewing Disk Storage Information**

Query the V_MONITOR.DISK_STORAGE system table for disk storage information on each database node. For more information, see Using System Tables and Altering Location Use. The V_MONITOR.DISK_STORAGE system table includes a CATALOG annotation, indicating that the location is used to store catalog files.

>`Note: You cannot add or remove a catalog storage location. Vertica creates and manages this storage location internally, and the area exists in the same location on each cluster node.`

**Viewing Location Labels**

Three system tables have information about storage location labels in their location_labels columns:
- storage_containers
- storage_locations
- partitions

Use a query such as the following for relevant columns of the storage_containers system table:

```sql
VMART=> select node_name, projection_name, location_label from v_monitor.storage_containers;
node_name | projection_name | location_label
-----------|----------------|----------------
v_vmart_node0001 | states_p | 
v_vmart_node0001 | states_p | 
v_vmart_node0001 | t1_b1 | 
v_vmart_node0001 | newstates_b0 | FAST3
v_vmart_node0001 | newstates_b0 | FAST3
v_vmart_node0001 | newstates_b1 | FAST3
v_vmart_node0001 | newstates_b1 | FAST3

Use a query such as the following for columns of the v_catalog.storage_locations system table:

```sql
VMart=> select node_name, location_path, location_usage, location_label from storage_locations;
node_name | location_path | location_usage | location_label
-----------|---------------|----------------|----------------
v_vmart_node0001 | /home/dbadmin/VMart/v_vmart_node0001_data | DATA,TEMP | 
v_vmart_node0001 | /home/dbadmin/SSD/schemas | DATA | 
v_vmart_node0001 | /home/dbadmin/SSD/tables | DATA | SSD
v_vmart_node0001 | /home/dbadmin/SSD/schemas | DATA | Schema
v_vmart_node0002 | /home/dbadmin/VMart/v_vmart_node0002_data | DATA,TEMP | 
v_vmart_node0002 | /home/dbadmin/SSD/tables | DATA | 
v_vmart_node0002 | /home/dbadmin/SSD/schemas | DATA | 
v_vmart_node0003 | /home/dbadmin/VMart/v_vmart_node0003_data | DATA,TEMP | 
v_vmart_node0003 | /home/dbadmin/SSD/tables | DATA | 
v_vmart_node0003 | /home/dbadmin/SSD/schemas | DATA | 

(10 rows)
```

Use a query such as the following for columns of the v_monitor.partitions system table:

```sql
VMART=> select partition_key, projection_name, location_label from v_monitor.partitions;
partition_key | projection_name | location_label
----------------|----------------|----------------
NH | states_b0 | FAST3
MA | states_b0 | FAST3
VT | states_b1 | FAST3
ME | states_b1 | FAST3
CT | states_b1 | FAST3
```

Vertica Analytic Database (9.0.x)
Viewing Storage Tiers

Query the `storage_tiers` system table to see both the labeled and unlabeled storage containers and information about them:

```
VMart=> select * from v_monitor.storage_tiers;
location_label | node_count | location_count | ros_container_count | total_occupied_size
-----------------|------------|----------------|---------------------|----------------------
                 |            |                |                     |                      
SSD              | 1          | 1              | 17                  | 297039391            
Schema           | 1          | 1              | 9                   | 1506                 
(3 rows)
```

Viewing Storage Policies

Query the `storage_policies` system table to view the current storage policy in place.

```
VMART=> select * from v_monitor.storage_policies;
schema_name | object_name | policy_details | location_label
--------------|-------------|----------------|------------------
public       | Schema      | F4             |                  
(2 rows)
```

Creating Storage Locations

You can add and configure storage locations (other than the required defaults) to provide storage for these purposes:

- Isolating execution engine temporary files from data files.
- Creating labeled locations to use in storage policies.
- Creating storage locations based on predicted or measured access patterns.
- Creating USER storage locations for specific users or user groups.

**Important:** While no technical issue prevents you from using `CREATE LOCATION` to add one or more Network File System (NFS) storage locations, Vertica does not support
NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify ALL NODES SHARED.

You can add a new storage location from one node to another node or from a single node to all cluster nodes. However, do not use a shared directory on one node for other cluster nodes to access.

Planning Storage Locations

Adding a storage location requires minimal planning:

1. Verify that the directory you plan to use for a storage location destination is an empty directory with write permissions for the Vertica process.

2. Plan the labels to use if you want to label the location as you create it.

3. Determine the type of information to store in the storage location:
   - DATA — Persistent data and temp table data.
   - TEMP — Temporary files that are generated and dumped to disk, such as those generated by sort, group by, join, and so on.
   - DATA,TEMP — Both data and temp files (the default).
   - USER — Gives access to non-dbadmin users so they can use the storage location after being granted read or write privileges. You cannot assign this location type for use in a storage policy.

Tip: Storing temp and data files in different storage locations is advantageous because the two types of data have different disk I/O access patterns. Temp files are distributed across locations based on available storage space. However, data files can be stored on different storage locations, based on storage policy, to reflect predicted or measured access patterns.

If you plan to place storage locations on HDFS, see Using HDFS Storage Locations in Integrating with Apache Hadoop for additional requirements.
Creating Unlabeled Local Storage Locations

This example shows a three-node cluster, each with a vertica/SSD directory for storage.

On each node in the cluster, create a directory where the node stores its data. For example:

$ mkdir /home/dbadmin/vertica/SSD

Vertica recommends that you create the same directory path on each node. Use this path when creating a storage location.

Use the CREATE LOCATION statement to add a storage location. Specify the following information:

- The path on the node where Vertica stores the data.
- The node where the location is available, or ALL NODES.

**Important:** Specify the node or use the ALL NODES keyword. Otherwise, the statement creates the storage locations on all nodes in the cluster in a single transaction.

- The type of information to be stored.

For user access (non-dbadmin users), you must create the storage location with the USER usage type. You cannot change an existing storage location to have USER access. After you create a USER storage location, you can grant one or more users access to it. User areas can store only data files, not temp files. You cannot assign a USER storage location to a storage policy.

The following example shows how to add a location available on all nodes to store only data:
The following example shows how to add a location that is available on node v_vmart_node0001 to store data and temporary files:

```
CREATE LOCATION '/home/dbadmin/vertica/SSD/' NODE 'v_vmart_node0001';
```

Suppose you are using a storage location for data files and want to create ranked storage locations. In this ranking, columns are stored on different disks based on their measured performance. To create ranked storage locations, see the following sections:

1. [Measuring Storage Performance](#)
2. [Setting Storage Performance](#)

**Note:** After you create a storage location, you can alter the type of information it stores, with some restrictions. See [Altering Location Use](#).

### Storage Location Subdirectories

You cannot create a storage location in a subdirectory of an existing location. Doing so results in an error similar to the following:

```
=> CREATE LOCATION '/tmp/myloc' ALL NODES USAGE 'TEMP';
CREATE LOCATION
=> CREATE LOCATION '/tmp/myloc/ssd' ALL NODES USAGE 'TEMP';
ERROR 5615: Location [/tmp/myloc/ssd] conflicts with existing location [/tmp/myloc] on node v_vmart_node0001
```

### Creating Labeled Storage Locations

You can add a storage location with a descriptive label using the CREATE LOCATION statement's LABEL keyword. You use labeled locations to set up storage policies for your site. See [Creating Storage Policies](#).

This example shows how to create a storage location on v_mart_node0002 with the label SSD:

```
=> CREATE LOCATION '/home/dbadmin/SSD/schemas' NODE 'v_mart_node0002' USAGE 'DATA' LABEL 'SSD';
```

This example shows how to create a storage location on all nodes. Specifying the ALL NODES keyword adds the storage location to all nodes in a single transaction:
CREATE LOCATION '/home/dbadmin/SSD/schemas' ALL NODES USAGE 'DATA' LABEL 'SSD';

The new storage location is listed in the v_monitor.disk_storage system table:

=> SELECT * FROM v_monitor.disk_storage;

<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmart_node002</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage_path</td>
<td>/home/dbadmin/SSD/schemas</td>
</tr>
<tr>
<td>storage_usage</td>
<td>DATA</td>
</tr>
<tr>
<td>rank</td>
<td>0</td>
</tr>
<tr>
<td>throughput</td>
<td>0</td>
</tr>
<tr>
<td>latency</td>
<td>0</td>
</tr>
<tr>
<td>storage_status</td>
<td>Active</td>
</tr>
<tr>
<td>disk_block_size_bytes</td>
<td>4096</td>
</tr>
<tr>
<td>disk_space_used_blocks</td>
<td>1549437</td>
</tr>
<tr>
<td>disk_space_used_mb</td>
<td>6053</td>
</tr>
<tr>
<td>disk_space_free_blocks</td>
<td>13380004</td>
</tr>
<tr>
<td>disk_space_free_mb</td>
<td>52265</td>
</tr>
<tr>
<td>disk_space_free_percent</td>
<td>89%</td>
</tr>
</tbody>
</table>

Creating a Storage Location for USER Access

You can create USER storage locations for a non-dbadmin user to access the storage location after being granted appropriate privileges.

The following example shows how to create a storage location, BillyBobStore, with a USER usage argument, on node v_mcdb_node0007:

=> CREATE LOCATION '/home/dbadmin/UserStorage/BillyBobStore' NODE 'v_mcdb_node0007' USAGE 'USER';

The following example shows how to grant a user BillyBob read and write permissions to the BillyBobStore location:

=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BillyBobStore' TO BillyBob;

For more information about configuring user privileges, see Managing Users and Privileges in the Administrator's Guide and the GRANT (Storage Location) and REVOKE (Storage Location) functions in the SQL Reference Manual.
Altering Location Use

You can make changes to the type of files that Vertica stores at a storage location. To modify a storage location, use the `ALTER_LOCATION_USE` function. Normally you use labels only for DATA storage locations, not TEMP.

This example shows how to alter the storage location on v_vmartdb_node0004 to store only data files:

```sql
=> SELECT ALTER_LOCATION_USE('/thirdVerticaStorageLocation/', 'v_vmartdb_node0004', 'DATA');
```

Altering HDFS Storage Locations

When altering an HDFS storage location, you must make the change for each node in the Vertica cluster. You can make the change on all nodes at the same time by specifying a node value of '', as in the following example:

```sql
=> SELECT ALTER_LOCATION_USE('hdfs:///user/dbadmin/v_vmart', '', 'TEMP');
```

USER Storage Location Restrictions

You cannot change a storage location from a USER usage type if you created the location that way, or to a USER type if you did not. You can change a USER storage location to specify DATA (storing TEMP files is not supported). However, doing so does not affect the primary objective of a USER storage location, to be accessible by non-dbadmin users with assigned privileges.

Effects of Altering Storage Location Use

Before altering a storage location use type, be aware that at least one location must remain for storing data and temp files on a node. You can store data and temp files in the same, or separate, storage locations.

Altering an existing storage location has the following effects:
<table>
<thead>
<tr>
<th>Alter use from:</th>
<th>To store only:</th>
<th>Has this effect:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp and data files (or data only)</td>
<td>Temp files</td>
<td>Data content is eventually merged out through ATM, per its policies. You can also merge out data from the storage location manually using DO_TM_TASK. The location stores only temp files from that point forward.</td>
</tr>
<tr>
<td>Temp and data files (or temp only)</td>
<td>Data files</td>
<td>Vertica continues to run all statements that use temp files (such as queries and loads). Subsequent statements no longer use the changed storage location for temp files, and the location stores only data files from that point forward.</td>
</tr>
</tbody>
</table>

**Altering Location Labels**

You can change the label for a storage location in the following ways:

- Add a label to an unlabeled storage location.
- Change an existing label.
- Remove a label.

You can perform this operation on individual nodes (by naming them) or cluster-wide (by specifying an empty string for the node).

**Important:** You can label an existing storage location that already contains data. However, including the labeled location in one or more storage policies can cause the ATM to relocate existing data. This situation occurs if the ATM determines that data stored on a labeled location does not comply with a storage policy.

**Adding a Location Label**

To alter a location label, use the ALTER_LOCATION_LABEL function. The following figure illustrates an unlabeled location that exists on three nodes:
This example shows how to add a location label, 'SSD', to the existing storage locations on all nodes (' '):

```
=> SELECT ALTER_LOCATION_LABEL('/home/dbadmin/vertica/SSD', '', 'SSD');
```

### Removing a Location Label

You cannot remove a location label if both of the following conditions exist:

- The name being removed is used in a storage policy
- The location from which you are removing the label is the last available storage for its associated objects

The following example removes the SSD label for the storage location on all nodes by specifying empty strings (' ') for both node and location_label parameters:

```
VMART=> select alter_location_label('/home/dbadmin/SSD/tables','','');
 alter_location_label
-------------------------------
/home/dbadmin/SSD/tables label changed.
(1 row)
```

### Effects of Altering a Location Label

Altering a location label has the following effects:
<table>
<thead>
<tr>
<th>Alter label</th>
<th>To:</th>
<th>Has this effect:</th>
</tr>
</thead>
<tbody>
<tr>
<td>No name</td>
<td>New label</td>
<td>Lets you use the labeled storage within a storage policy. See note above regarding data storage being moved to other locations if you add a label to a storage location with existing data.</td>
</tr>
<tr>
<td>Existing name</td>
<td>New name</td>
<td>You can use the new label in a storage policy. If the existing name is used in a storage policy, you cannot change the label.</td>
</tr>
<tr>
<td>Existing name</td>
<td>No name</td>
<td>You cannot use an unlabeled storage in a storage policy. If the existing name is used in a storage policy, you cannot remove the label.</td>
</tr>
</tbody>
</table>

**Creating Storage Policies**

You create a storage policy to associate a database object with a labeled storage location. To do so, use the `SET_OBJECT_STORAGE_POLICY` function. When a storage policy exists, Vertica uses the labeled location as the default storage location for the object data. Storage policies let you determine where to store your critical data. For example, suppose you create a storage location with the label SSD representing the fastest available storage on the cluster nodes. You can then create storage policies to associate tables with that labeled location. One storage policy can exist per database object.

**Note:** You cannot include temporary files in storage policies. Storage policies are for use only with data files on storage locations for DATA. Storage policies are not valid for USER locations.

You can create a storage policy for any database object (database, schemas, tables, and partition ranges). Each time data is loaded and updated, Vertica checks to see whether the object has a storage policy. If it does, Vertica automatically uses the labeled storage location. If no storage policy exists for an object, or its parent entities, data storage processing continues using standard storage algorithms on available storage locations. If all storage locations are labeled, Vertica uses one of them.

Creating one or more storage policies does not require that policies exist for all database objects. A site can support objects with or without storage policies. You can add storage policies for a discrete set of priority objects, while letting other objects exist without a policy, so they use available storage.
Creating Policies Based on Storage Performance

You can measure the performance of any disk storage location (see Measuring Storage Performance). Then, using the performance measurements, set the storage location performance. Vertica uses the performance measurements you set to rank its storage locations and, through ranking, to determine which key projection columns to store on higher performing locations, as described in Setting Storage Performance.

If you have already set the performance of your site's storage locations, and decide to use storage policies, any storage location with an associated policy has a higher priority than the storage ranking setting.

Storage Levels and Priorities

Vertica assigns storage levels to database objects. The database is the highest storage level (because nothing exists above the database level), and partition min_ and max_key ranges are considered the lowest level objects. In addition to storage levels, storage priorities exist. The lower the storage level of an object, the higher its storage priority.

Consider this example of database objects, listed in storage level order, with the highest level, Sales database, first:

<table>
<thead>
<tr>
<th>Object</th>
<th>Storage Level</th>
<th>Storage Policy</th>
<th>Storage Priority</th>
<th>Labeled Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales (database)</td>
<td>Highest</td>
<td>YES</td>
<td>Lower</td>
<td>STANDARD</td>
</tr>
<tr>
<td>Region (schema)</td>
<td>Medium</td>
<td>NO</td>
<td>Medium</td>
<td>N/A</td>
</tr>
<tr>
<td>Income (table)</td>
<td>Lower</td>
<td>YES</td>
<td>Higher/highest</td>
<td>FAST</td>
</tr>
<tr>
<td>Month (partitions)</td>
<td>Lowest</td>
<td>NO</td>
<td>Highest</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Storage policies exist for the database and table objects, with default storage on the locations STANDARD and FAST, respectively.

In this case, when TM operations occur, such as moveout and mergeout, table data has the highest priority. The TM moves data from WOS to ROS to the FAST labeled location.

Any schema data changes are prioritized after table data. Because the Region schema has no storage policy, Vertica searches up the storage levels for a policy. In this case, that is the Sales database itself. If a database storage policy is in effect, Region schema data is moved from
WOS to ROS to the STANDARD storage location, using its parent object's default storage location.

If the Sales database object had no storage policy, the TM operations would use existing storage locations and mechanisms.

Using the SET_OBJECT_STORAGE_POLICY Function

To set a storage policy, use the SET_OBJECT_STORAGE_POLICY function.

This example shows how to set a storage policy for the table test to use the storage labeled SSD as its default location:

```sql
=> select set_object_storage_policy('test', 'ssd', true);
set_object_storage_policy
-----------------------------------------------
Object storage policy set.
Task: moving storages
(Table: public.test) (Projection: public.test_b0)
(Table: public.test) (Projection: public.test_b1)

(1 row)
```

You can query existing storage policies, listed in the location_label column of the v_monitor.storage_containers system table:

```
VMART=> select node_name, projection_name, storage_type, location_label from v_monitor.storage_containers;
           node_name  |    projection_name    |       storage_type       | location_label
-------------------------------------------------------------
 v_vmart_node0001 |      states_p      |             SSD          |
 v_vmart_node0001 |      states_p      |             SSD          |
 v_vmart_node0001 |       t1_b1       |             SSD          |
 v_vmart_node0001 | newstates_b0      |       LEVEL3            |
 v_vmart_node0001 | newstates_b0      |       LEVEL3            |
 v_vmart_node0001 | newstates_b1      |       LEVEL3            |
 v_vmart_node0001 | newstates_b1      |       LEVEL3            |
 v_vmart_node0001 | newstates_b1      |       LEVEL3            |
 v_vmart_node0001 | newstates_b1      |       LEVEL3            |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
 v_vmart_node0001 | states_p_v1_node0001 |   LEVEL3                |
```

```
You can use storage policies to move older data to less-expensive storage locations while keeping it available for queries. See Creating Storage Policies for Low-Priority Data.

Effects of Creating Storage Policies

Creating storage policies has the following effects:

<table>
<thead>
<tr>
<th>Create policy for</th>
<th>Storage effect:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>The highest storage level and the lowest storage priority. This is the default policy when no lower-level or higher priority policies exist. At storage time, Vertica uses the database policy for all objects without storage policies.</td>
</tr>
<tr>
<td>Schema</td>
<td>The mid-level storage, also with a medium priority, compared to lower storage level objects. If a table’s schema has no policy, the TM searches the next higher level, the database, using that policy, if it exists. If it does not, the TM uses existing storage mechanisms.</td>
</tr>
<tr>
<td>Table</td>
<td>A lower storage level than a schema, with the highest storage priority, if no policy exists for the table's partition key ranges. If a table has no storage policy, Vertica checks the next higher storage level (the schema) for a policy and uses that. If the schema has no policy, it checks the next higher level, the database, and uses that. If no database policy exists, the TM uses existing storage mechanisms.</td>
</tr>
<tr>
<td>Partition</td>
<td>The lowest level policy that can be in effect. During storage processing, partition key ranges with a policy have the highest priority. If no policy exists, the parent table is checked, and so on as described for the other database objects.</td>
</tr>
<tr>
<td>min_key and max_</td>
<td></td>
</tr>
<tr>
<td>key ranges</td>
<td></td>
</tr>
</tbody>
</table>

Creating Storage Policies for Low-Priority Data

If some of your data is in a partitioned table, you can move less-queried partitions to less-expensive storage such as HDFS. The data is still accessible in queries, just at a slower speed. In this scenario, the faster storage is often referred to as "hot storage," and the slower storage is referred to as "cold storage."
Suppose you have a table named messages (containing social-media messages) that is partitioned by the year and month of the message's timestamp. You can list the partitions in the table by querying the PARTITIONS system table.

```sql
=> SELECT partition_key, projection_name, node_name, location_label FROM partitions
   ORDER BY partition_key;
```

<table>
<thead>
<tr>
<th>partition_key</th>
<th>projection_name</th>
<th>node_name</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>201309</td>
<td>messages_b1</td>
<td>v_vmart_node001</td>
<td></td>
</tr>
<tr>
<td>201309</td>
<td>messages_b0</td>
<td>v_vmart_node003</td>
<td></td>
</tr>
<tr>
<td>201309</td>
<td>messages_b1</td>
<td>v_vmart_node002</td>
<td></td>
</tr>
<tr>
<td>201309</td>
<td>messages_b1</td>
<td>v_vmart_node003</td>
<td></td>
</tr>
<tr>
<td>201309</td>
<td>messages_b0</td>
<td>v_vmart_node001</td>
<td></td>
</tr>
<tr>
<td>201309</td>
<td>messages_b0</td>
<td>v_vmart_node002</td>
<td></td>
</tr>
<tr>
<td>201310</td>
<td>messages_b0</td>
<td>v_vmart_node002</td>
<td></td>
</tr>
<tr>
<td>201310</td>
<td>messages_b1</td>
<td>v_vmart_node003</td>
<td></td>
</tr>
<tr>
<td>201310</td>
<td>messages_b0</td>
<td>v_vmart_node001</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201405</td>
<td>messages_b0</td>
<td>v_vmart_node002</td>
<td></td>
</tr>
<tr>
<td>201405</td>
<td>messages_b1</td>
<td>v_vmart_node003</td>
<td></td>
</tr>
<tr>
<td>201405</td>
<td>messages_b1</td>
<td>v_vmart_node001</td>
<td></td>
</tr>
<tr>
<td>201405</td>
<td>messages_b0</td>
<td>v_vmart_node001</td>
<td></td>
</tr>
<tr>
<td>(54 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next, suppose you find that most queries on this table access only the latest month or two of data. You might decide to move the older data to cold storage in an HDFS-based storage location. After you move the data, it is still available for queries, but with lower query performance.

To move partitions to the HDFS storage location, supply the lowest and highest partition key values to be moved in the `SET_OBJECT_STORAGE_POLICY` function call. The following example shows how to move data between two dates. In this example:

- The partition key value 201309 represents September 2013.
- The partition key value 201403 represents March 2014.
- The name, coldstorage, is the label of the HDFS-based storage location.
- The final argument, which is optional, is `true`, meaning that the function does not return until the move is complete. By default the function returns immediately and the data is moved when the Tuple Mover next runs. When data is old, however, the Tuple Mover runs less frequently, which would delay recovering the original storage space.

```sql
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'coldstorage', '201309', '201403', 'true');
```

The partitions within the specified range are moved to the HDFS storage location labeled coldstorage the next time the Tuple Mover runs. This location name now displays in the PARTITIONS system table's location_label column.
After your initial data move, you can move additional data to the HDFS storage location periodically. You can move individual partitions or a range of partitions from the "hot" storage to the "cold" storage location using the same method:

```sql
=> SELECT partition_key, projection_name, node_name, location_label
    FROM partitions ORDER BY partition_key;
```

<table>
<thead>
<tr>
<th>partition_key</th>
<th>projection_name</th>
<th>node_name</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>201309</td>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201309</td>
<td>messages_b1</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201309</td>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201403</td>
<td>messages_b0</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201404</td>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201404</td>
<td>messages_b1</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201404</td>
<td>messages_b0</td>
<td>v_vmart_node0003</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201404</td>
<td>messages_b1</td>
<td>v_vmart_node0003</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b1</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b0</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b1</td>
<td>v_vmart_node0003</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>201405</td>
<td>messages_b1</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
</tbody>
</table>
(54 rows)

```sql
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'coldstorage', '201404', '201404', 'true');
```

```sql
=> SELECT projection_name, node_name, location_label
    FROM PARTITIONS WHERE PARTITION_KEY = '201404';
```

<table>
<thead>
<tr>
<th>projection_name</th>
<th>node_name</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>messages_b0</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>messages_b0</td>
<td>v_vmart_node0003</td>
<td>coldstorage</td>
</tr>
<tr>
<td>messages_b1</td>
<td>v_vmart_node0003</td>
<td>coldstorage</td>
</tr>
<tr>
<td>messages_b0</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
<tr>
<td>messages_b1</td>
<td>v_vmart_node0002</td>
<td>coldstorage</td>
</tr>
<tr>
<td>messages_b1</td>
<td>v_vmart_node0001</td>
<td>coldstorage</td>
</tr>
</tbody>
</table>
(6 rows)

### Moving Partitions to a Table Stored on HDFS

Another method of moving partitions from hot storage to cold storage is to move the partitions' data to a separate table in the other storage location. This method breaks the data into two tables, one containing hot data and the other containing cold data. Use this method if you want to prevent queries from inadvertently accessing data stored in cold storage. To query the older data, you must explicitly query the cold table.

To move partitions:
1. Create a new table whose schema matches that of the existing partitioned table.

2. Set the storage policy of the new table to use the HDFS-based storage location.

3. Use the `MOVE_PARTITIONS_TO_TABLE` function to move a range of partitions from the hot table to the cold table. The partitions migrate when the Tuple Mover next runs.

The following example demonstrates these steps. You first create a table named `cold_messages`. You then assign it the HDFS-based storage location named `coldstorage`, and, finally, move a range of partitions.

```sql
=> CREATE TABLE cold_messages LIKE messages INCLUDING PROJECTIONS;
=> SELECT SET_OBJECT_STORAGE_POLICY('cold_messages', 'coldstorage');
=> SELECT MOVE_PARTITIONS_TO_TABLE('messages','201309','201403','cold_messages');
```

### Moving Data Storage Locations

You can use the `SET_OBJECT_STORAGE_POLICY` function to move data storage from an existing location (whether labeled or not) to another labeled location. You can use this function to accomplish two tasks:

1. Create or update a storage policy:
   - Create new storage policy for the object
   - Update an existing policy to the target labeled location.

2. Move all existing data for the specified objects to the target storage location.

Vertica moves the existing data the next time the Tuple Mover runs. Alternatively, you can enforce the data move to occur in the current transaction using the function's `enforce_storage_move` parameter. Consider setting this parameter to `true` if the Tuple Mover will not run on this data for a while, such as if the data is old.

Before actually moving the object to the target storage location, Vertica calculates the required storage and checks available space at the target. If there is insufficient free space, the function generates an error and stops execution. It does not attempt to find sufficient storage at another location.

**Important:** You should check available space on the new target location before starting a data move. However, be aware that checking does not guarantee that this space remains available during move execution. Checking target space does prevent attempts to move...
any data, if insufficient space is available.

## Moving Data Storage While Setting a Storage Policy

The following example shows how to use `SET_OBJECT_STORAGE POLICY` to set a storage policy for the table object `states`. Then, you can move the table's existing stored data to the labeled location, SSD. You force the move to occur during the function transaction by specifying the last parameter as true:

```sql
=> select set_object_storage_policy('states', 'SSD', 'true');
```

---

### Note:
Moving an object's current storage to a new target is a cluster-wide operation, so a failure on any node results in a warning message. The function then attempts to continue executing on other cluster nodes.

You can view the storage policies that are in effect:

```sql
=> select * from storage_policies;
```

<table>
<thead>
<tr>
<th>schema_name</th>
<th>object_name</th>
<th>policy_details</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>states</td>
<td>Table</td>
<td>SSD</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Effects of Moving a Storage Location

Moving an object from one labeled storage location to another has the following effects:

<table>
<thead>
<tr>
<th>Object type:</th>
<th>Effect:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema or table</td>
<td>If data storage exists, moves data from source to target destination.</td>
</tr>
<tr>
<td></td>
<td>Source data can reside on a labeled or unlabeled storage location, but will be moved to specified labeled location.</td>
</tr>
<tr>
<td></td>
<td>Cluster nodes unavailable when existing data is copied are updated by the TM when they rejoin the cluster. Alternately, you can enforce a data</td>
</tr>
</tbody>
</table>
Object type: | Effect:
---|---
| move as part of the function transaction, by specifying the last parameter as true. If a storage policy was in effect, the default storage location changes from the source to target location. This change affects all future TM operations, such as moveout and mergeout activities.

Table with specified partition min-keys and max_keys | Sets a policy or moves existing data only for the key_min and key_max ranges. Separate partition key ranges can have different storage policies from other ranges or the parent table.

## Clearing Storage Policies

You can clear a storage policy by object name after you have defined storage policies. To see existing policies, query the storage_policies system table, described in Viewing Storage Locations and Policies.

To clear a storage policy, use the `CLEAR_OBJECT_STORAGE_POLICY` function, specifying the object name associated with the labeled location:

```
=> select clear_object_storage_policy('lineorder');
clear_object_storage_policy
-------------------------------
Default storage policy cleared.
(1 row)
```

Existing data is moved to the parent storage policy's location, or the default storage location if there is no parent policy. Typically, the move occurs the next time the Tuple Mover runs. Alternatively, you can force the data move to occur in the current transaction by setting the optional `enforce_storage_move` parameter to `true`.

You can also use the `ENFORCE_OBJECT_STORAGE_POLICY` meta-function to trigger the move for all storage locations at once. Using this function equates to setting `enforce_storage_move`.
Speeding Up Data Migration

After you clear the storage policy, the Tuple Mover eventually migrates the object's data from the storage location to the database's default storage location. The TM moves the data when it performs a move-storage operation. This operation runs infrequently at low priority. Therefore, it might be some time before the data migrates out of the storage location.

You can speed up the data-migration process by:

1. Calling the `RETIRE_LOCATION` function to retire the storage location on each host that defines it.

2. Calling the `MOVE RETIRED LOCATION DATA` function to move the location's data to the database's default storage location.

3. Calling the `RESTORE LOCATION` function to restore the location on each host that defines it. You must perform this step because you cannot drop retired storage locations.

The following example demonstrates clearing the object storage policy of a table stored on HDFS, then performing the steps to move the data off of the location. (See Using HDFS Storage Locations for additional background.)

```sql
=> SELECT * FROM storage_policies;
schema_name | object_name | policy_details | location_label
--------------|-------------|----------------|----------------|
public        | test        | Table          | hdfs2
(1 row)

=> SELECT clear_object_storage_policy('test');
clear_object_storage_policy
----------------------------------------
Object storage policy cleared.
(1 row)

=> SELECT retire_location('hdfs:///user/dbadmin/v_vmart_node0001',
  'v_vmart_node0001');
  retire_location
-------------------------------
hdfs:///user/dbadmin/v_vmart_node0001 retired.
(1 row)

=> SELECT retire_location('hdfs:///user/dbadmin/v_vmart_node0002',
  'v_vmart_node0002');
  retire_location
-------------------------------
hdfs:///user/dbadmin/v_vmart_node0002 retired.
(1 row)

=> SELECT retire_location('hdfs:///user/dbadmin/v_vmart_node0003',
  'v_vmart_node0003');
  retire_location
-------------------------------
hdfs:///user/dbadmin/v_vmart_node0003 retired.
(1 row)
```
'v_vmart_node0003');

  retire_location
---------------------------------------------------------------------
hdfs:///user/dbadmin/v_vmart_node0003 retired.
(1 row)

=> SELECT node_name, projection_name, location_label, total_row_count FROM V_MONITOR.STORAGE_CONTAINERS WHERE projection_name ILIKE 'test%';

<table>
<thead>
<tr>
<th>node_name</th>
<th>projection_name</th>
<th>location_label</th>
<th>total_row_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>test_b1</td>
<td>hdfs2</td>
<td>333631</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>test_b0</td>
<td>hdfs2</td>
<td>332233</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>test_b1</td>
<td>hdfs2</td>
<td>332233</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>test_b0</td>
<td>hdfs2</td>
<td>334136</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>test_b1</td>
<td>hdfs2</td>
<td>334136</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>test_b0</td>
<td>hdfs2</td>
<td>333631</td>
</tr>
</tbody>
</table>

(6 rows)

=> SELECT move_retired_location_data();

  move_retired_location_data
---------------------------------------------------------------------
Move data off retired storage locations done

(1 row)

=> SELECT node_name, projection_name, location_label, total_row_count FROM V_MONITOR.STORAGE_CONTAINERS WHERE projection_name ILIKE 'test%';

<table>
<thead>
<tr>
<th>node_name</th>
<th>projection_name</th>
<th>location_label</th>
<th>total_row_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>test_b0</td>
<td></td>
<td>332233</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>test_b1</td>
<td></td>
<td>333631</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>test_b0</td>
<td></td>
<td>334136</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>test_b1</td>
<td></td>
<td>332233</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>test_b0</td>
<td></td>
<td>333631</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>test_b1</td>
<td></td>
<td>334136</td>
</tr>
</tbody>
</table>

(6 rows)

=> SELECT restore_location('hdfs:///user/dbadmin/v_vmart_node0001', 'v_vmart_node0001');

  restore_location
---------------------------------------------------------------------
hdfs:///user/dbadmin/v_vmart_node0001 restored.
(1 row)

=> SELECT restore_location('hdfs:///user/dbadmin/v_vmart_node0002', 'v_vmart_node0002');

  restore_location
---------------------------------------------------------------------
hdfs:///user/dbadmin/v_vmart_node0002 restored.
(1 row)

=> SELECT restore_location('hdfs:///user/dbadmin/v_vmart_node0003', 'v_vmart_node0003');

  restore_location
---------------------------------------------------------------------
hdfs:///user/dbadmin/v_vmart_node0003 restored.
(1 row)
Effects on Same-Name Storage Policies

The effects of clearing a storage policy depend on which policy you clear.

For example, consider the following storage configuration. The table `lineorder` has a storage policy for default storage to the location label F2. The table’s partition ranges, also `lineorder` objects, have storage policies for other default storage locations:

```
=> select * from v_monitor.storage_policies;
  schema_name | object_name | policy_details | location_label
---------------|-------------|----------------|----------------
  public       | public      | Schema         | F4
  public       | lineorder   | Table          | F2
  public       | lineorder   | Partition [0, 0] | F1
  public       | lineorder   | Partition [1, 1] | F2
  public       | lineorder   | Partition [2, 2] | F4
  public       | lineorder   | Partition [3, 3] | M1
(7 rows)
```

For this example, clearing the storage policy for an object named `lineorder`, removes the policy for the table, while retaining storage policies for its partitions, which have their own policies.

The function determines which `lineorder` object policy to clear because no partition range values are specified in the function call:

```
=> select clear_object_storage_policy('lineorder');
  clear_object_storage_policy
-------------------------------
  Default storage policy cleared.
(1 row)
release=> select * from v_monitor.storage_policies;
  schema_name | object_name | policy_details | location_label
---------------|-------------|----------------|----------------
  public       | lineorder   | Partition [0, 0] | F1
  public       | lineorder   | Partition [1, 1] | F2
  public       | lineorder   | Partition [2, 2] | F4
  public       | lineorder   | Partition [3, 3] | M1
(6 rows)
```

Further, using a partition key range with the `lineorder` object name clears the storage policy for only the specified partition range(s). The storage policy for the parent table objects, and other partition ranges persist:

```
=> select clear_object_storage_policy ('lineorder','0','3');
  clear_object_storage_policy
-------------------------------
```

```
Default storage policy cleared.
(1 row)
release=> select * from storage_policies;

<table>
<thead>
<tr>
<th>schema_name</th>
<th>object_name</th>
<th>policy_details</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Schema</td>
<td>F4</td>
<td>public</td>
</tr>
<tr>
<td>public</td>
<td>lineorder</td>
<td>Table</td>
<td>public</td>
</tr>
</tbody>
</table>

(2 rows)

**Measuring Storage Performance**

Vertica lets you measure disk I/O performance on any storage location at your site. You can use the returned measurements to set performance, which automatically provides rank. Depending on your storage needs, you can also use performance to determine the storage locations needed for critical data as part of your site’s storage policies. Storage performance measurements apply only to data storage locations, not temporary storage locations.

Measuring storage location performance calculates the time it takes to read and write 1 MB of data from the disk, which equates to:

\[
\text{IO time} = \text{time to read/write 1MB} + \text{time to seek} = \frac{1}{\text{throughput}} + \frac{1}{\text{Latency}}
\]

- Throughput is the average throughput of sequential reads/writes (expressed in megabytes per second).
- Latency is for random reads only in seeks (units in seeks per second).

Thus, the I/O time of a faster storage location is less than that of a slower storage location.

**Note:** Measuring storage location performance requires extensive disk I/O, which is a resource-intensive operation. Consider starting this operation when fewer other operations are running.

Vertica gives you two ways to measure storage location performance, depending on whether the database is running. You can either:

- Measure performance on a running database.
- Measure performance before a cluster is set up.

Both methods return the throughput and latency for the storage location. Record or capture the throughput and latency information so you can use it to set the location performance (see [Setting Storage Performance](#)).
Measuring Performance on a Running Vertica Database

Use the `MEASURE_LOCATION_PERFORMANCE()` function to measure performance for a storage location when the database is running. This function has the following requirements:

- The storage path must already exist in the database.
- You need RAM*2 free space available in a storage location to measure its performance. For example, if you have 16 GB RAM, you need 32 GB of available disk space. If you do not have enough disk space, the function returns an error.

Use the system table `DISK_STORAGE` to obtain information about disk storage on each database node.

The following example shows how to measure the performance of a storage location on `v_vmartdb_node0004`:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/','v_vmartdb_node0004');
WARNING: measure_location_performance can take a long time. Please check logs for progress
measure_location_performance
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

Measuring Performance Before a Cluster Is Set Up

You can measure disk performance before setting up a cluster. This approach is useful when you want to verify that the disk is functioning within normal parameters. To perform this measurement, you must already have Vertica installed.

To measure disk performance, use the following command:

```
opt/vertica/bin/vertica -m <path to disk mount>
```

For example:

```
opt/vertica/bin/vertica -m /secondVerticaStorageLocation/node0004_data
```

Setting Storage Performance

You can use the measurements returned from the `MEASURE_LOCATION_PERFORMANCE` function as input values to the `SET_LOCATION_PERFORMANCE()` function.
Note: You must set the throughput and latency parameters of this function to 1 or more.

The following example shows how to set the performance of a storage location on v_vmardb_node0004, using values for this location returned from the `MEASURE_LOCATION_PERFORMANCE` function. Set the throughput to 122 MB/second and the latency to 140 seeks/second. `MEASURE_LOCATION_PERFORMANCE`

```sql
=> SELECT SET_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/','node2','122','140');
```

Sort Order Ranking by Location Performance Settings

After you set performance-data parameters, Vertica automatically uses performance data to rank storage locations whenever it stores projection columns.

Vertica stores columns included in the projection sort order on the fastest available storage locations. Columns not included in the projection sort order are stored on slower disks. Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000–1, starting with 1000 and decrementing by one per column.

Vertica then stores columns on disk from the highest ranking to the lowest ranking. It places highest-ranking columns on the fastest disks and the lowest-ranking columns on the slowest disks.

Using Location Performance Settings with Storage Policies

You initially measure location performance and set it in the Vertica database. Then, you can use the performance results to determine the fastest storage to use in your storage policies.
Set the locations with the highest performance as the default locations for critical data.

Use slower locations as default locations for older, or less-important data. Such slower locations may not require policies at all, if you do not want to specify default locations.

Vertica determines data storage as follows, depending on whether a storage policy exists:

<table>
<thead>
<tr>
<th>Storage Policy</th>
<th>Label</th>
<th>Number of Locations</th>
<th>Vertica Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Multiple</td>
<td>Uses ranking (as described), choosing a location from all locations that exist.</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Single</td>
<td>Uses that storage location exclusively.</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Multiple</td>
<td>Ranks storage (as described) among all same-name labeled locations.</td>
</tr>
</tbody>
</table>

## Retiring Storage Locations

To retire a storage location, use the `RETIRE_LOCATION()` function.

The following example shows how to retire a storage location on `v_vmartdb_node0004`:

```sql
=> SELECT RETIRE_LOCATION('/secondStoragelocation/' , 'v_vmartdb_node0004');
```

To retire a storage location on all nodes, use an empty string (""") for the second parameter.

Retiring a location prevents Vertica from storing data or temp files to it, but does not remove the actual location. Any data previously stored on the retired location is eventually merged out by the Automatic Tuple Mover (ATM) per its policies.

If you plan to drop the storage location after retiring it, you can expedite this by setting the optional `enforce_storage_move` parameter to `true`. This setting moves the data out of the storage location instead of waiting for the Tuple Mover, allowing you to drop the location immediately.

You can also use the `ENFORCE_OBJECT_STORAGE_POLICY` meta-function to trigger the move for all storage locations at once, and then drop the locations. This approach equates to setting `enforce_storage_move`. 
The following example shows how to retire a storage location on all nodes and prepares it for immediate drop:

```sql
=> SELECT RETIRE_LOCATION('/secondStorageLocation/', '', true);
```

**Note:** If the location used in a storage policy is the last available storage for its associated objects, you cannot retire it unless you set `enforce_storage_move` to true.

Data and temp files can be stored in one, or multiple separate, storage locations. For further information on dropping a location after retiring it, see [Dropping Storage Locations](#).

---

## Dropping Storage Locations

To drop a storage location, use the `DROP_LOCATION()` function. The following example shows how to drop a storage location on v_vmartdb_node0002 that was used to store temp files:

```sql
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/', 'v_vmartdb_node0002');
```

When you drop a storage location, the operation cascades to associated objects including any granted privileges to the storage.

**Caution:** Dropping a storage location is a permanent operation and cannot be undone. Subsequent queries on storage used for external table access fail with a COPY COMMAND FAILED message.

Because dropping a storage location cannot be undone, Vertica recommends that you first retire a storage location (see [Retiring Storage Locations](#)). Retiring a storage location before dropping it lets you verify that there will be no adverse effects on any data access. Additionally, you can restore a retired storage location (see [Restoring Retired Storage Locations](#)).

---

## Altering Storage Locations Before Dropping Them

You can drop only storage locations containing temp files. Thus, you must alter a storage location to the TEMP usage type before you can drop it. However, if data files still exist on the storage, Vertica prevents you from dropping the storage location. Deleting data files does not clear the storage location and can result in database corruption. To handle a storage area containing data files so that you can drop it, use one of these options:
- Manually merge out the data files.
- Wait for the ATM to merge out the data files automatically.
- Retire the location, and force the changes to take effect immediately.
- Manually drop partitions

### Dropping USER Storage Locations

Storage locations that you create with the USER usage type can contain only data files, not temp files. However, you can drop a USER location, regardless of any remaining data files. This behavior differs from that of a storage location not designated for USER access.

### Checking Location Properties

You can check the properties of a storage location, such as whether it is a USER location or is being used only for TEMP files, in the STORAGE_LOCATIONS system table. You can also use this table to verify that a location has been retired.

### Restoring Retired Storage Locations

You can restore a previously retired storage location that continues to be used in queries. After the location is restored, Vertica re-ranks the storage location and uses the restored location to process queries as determined by its rank.

Use the `RESTORE_LOCATION()` function to restore a retired storage location.

The following example shows how to restore a retired storage location on `v_vmartdb_node0004`:

```sql
=> SELECT RESTORE_LOCATION('/secondVerticaStorageLocation/', 'v_vmartdb_node0004');
```

To restore a storage location on all nodes, use an empty string ("") for the second parameter. The following example demonstrates creating, retiring, and restoring a location on all nodes:

```sql
=> create location '/tmp/ab1' all nodes usage 'TEMP';
CREATE LOCATION
=> select retire_location('/tmp/ab1', '');
```
retire_location
-------------------
/tmp/ab1 retired.
(1 row)

=> select * from storage_locations where location_path like '/tmp/ab1';

<table>
<thead>
<tr>
<th>location_id</th>
<th>node_name</th>
<th>location_path</th>
<th>location_usage</th>
<th>is_retired</th>
<th>location_label</th>
<th>rank</th>
<th>throughput</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v_vmart_node0001</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0002</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0003</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0004</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)

=> select restore_location(''/tmp/ab1'', '');

restore_location
-------------------
/tmp/ab1 restored.
(1 row)

=> select * from storage_locations where location_path like '/tmp/ab1';

<table>
<thead>
<tr>
<th>location_id</th>
<th>node_name</th>
<th>location_path</th>
<th>location_usage</th>
<th>is_retired</th>
<th>location_label</th>
<th>rank</th>
<th>throughput</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v_vmart_node0001</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0002</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0003</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_vmart_node0004</td>
<td>/tmp/ab1</td>
<td>TEMP</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)

The RESTORE_LOCATION() function restores the location only on the nodes where the location exists and is retired. The function does not propagate the storage location to nodes where that location did not previously exist.

Restoring on all nodes fails if the location has been dropped on any of them. If you have dropped the location on some nodes, you have two options:

- If you no longer want to use the dropped node, restore the location individually on each of the other nodes.

- Alternatively, you can re-create the location on the node where you dropped it. To do so, use CREATE LOCATION() or ADD_LOCATION(). After you have re-created the location, you can then restore it on all nodes.
The following example demonstrates the failure if you try to restore on nodes where you have dropped the location:

```sql
=> select retire_location('/tmp/ab1', '');
retire_location
-------------------------
/tmp/ab1 retired.
(1 row)

=> select drop_location('/tmp/ab1', 'v_vmart_node0002');
drop_location
-------------------------
/tmp/ab1 dropped.
(1 row)

=> select * from storage_locations where location_path ilike '/tmp/ab1';
location_id | node_name       | location_path | location_usage | is_retired | location_label | rank | throughput | latency
-------------+-----------------+---------------+----------------+------------+---------------+-----+------------+--------
45035996273736724 | v_vmart_node0001 | /tmp/ab1     | TEMP           | t         |               | 0   | 0         | 0      
45035996273736728 | v_vmart_node0003 | /tmp/ab1     | TEMP           | t         |               | 0   | 0         | 0      
45035996273736730 | v_vmart_node0004 | /tmp/ab1     | TEMP           | t         |               | 0   | 0         | 0      
(3 rows)

=> select restore_location('/tmp/ab1', '');
ERROR 2081: [/tmp/ab1] is not a valid storage location on node v_vmart_node0002
```
Analyzing Workloads

If queries perform suboptimally, use the Workload Analyzer (WLA) to get tuning recommendations for them and hints about optimizing database objects. WLA is a Vertica utility that analyzes system information in Vertica system tables.

WLA identifies the root causes of poor query performance through intelligent monitoring of query execution, workload history, resources, and configurations. It then returns a set of tuning recommendations based on statistics, system and data collector events, and database/table/projection design. Use these recommendations to tune query performance, quickly and easily.

You can run WLA in two ways:

- Call the Vertica function `ANALYZE_WORKLOAD`.
- Use the Management Console interface.

See Understanding WLA Triggering Conditions for the most common triggering conditions and recommendations.

Getting Tuning Recommendations

Call the function `ANALYZE_WORKLOAD` to get tuning recommendations for queries and database objects. The function arguments specify what events to analyze and when.

Setting Scope and Time Span

`ANALYZE_WORKLOAD`'s scope argument determines what to analyze:

<table>
<thead>
<tr>
<th>This argument...</th>
<th>Returns WLA recommendations for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>'' (empty string)</td>
<td>All database objects</td>
</tr>
<tr>
<td>Table name</td>
<td>A specific table</td>
</tr>
<tr>
<td>Schema name</td>
<td>All objects in the specified schema</td>
</tr>
</tbody>
</table>

The optional since-time argument specifies to return values from all in -scope events starting from since-time and continuing to the current system status. If you omit since-time, `ANALYZE_WORKLOAD` returns recommendations for events since the last recorded time.
that you called the function. You must explicitly cast the \texttt{since-time} string value to either \texttt{TIMESTAMP} or \texttt{TIMESTAMPTZ}.

The following examples show four ways to express the \texttt{since-time} argument with different formats. All queries return the same result for workloads on table \texttt{t1} since October 4, 2012:

```sql
=> SELECT ANALYZE_WORKLOAD('t1', TIMESTAMP '2012-10-04 11:18:15');
=> SELECT ANALYZE_WORKLOAD('t1', '2012-10-04 11:18:15'::TIMESTAMPTZ);
=> SELECT ANALYZE_WORKLOAD('t1', 'October 4, 2012'::TIMESTAMP);
=> SELECT ANALYZE_WORKLOAD('t1', '10-04-12'::TIMESTAMPTZ);
```

### Saving Function Results

Instead of analyzing events since a specific time, you can save results from \texttt{ANALYZE_WORKLOAD}, by setting the function's second argument to \texttt{true}. The default is \texttt{false}, and no results are saved. After saving function results, subsequent calls to \texttt{ANALYZE_WORKLOAD} analyze only events since you last saved returned data, and ignore all previous events.

For example, the following statement returns recommendations for all database objects in all schemas and records this analysis invocation.

```sql
=> SELECT ANALYZE_WORKLOAD('', true);
```

The next invocation of \texttt{ANALYZE_WORKLOAD} analyzes events from this point forward.
Observation Count and Time

The observation_count column returns an integer that represents the total number of events Workload Analyzer (WLA) observed for this tuning recommendation. In each case above, WLA is making its first recommendation. Null results in observation_time only mean that the recommendations are from the current system status instead of from a prior event.

Tuning Targets

The tuning_parameter column returns the object on which WLA recommends that you apply the tuning action. The parameter of release in the example above notifies the DBA to set a password for user release.

Tuning Recommendations and Costs

WLA's output returns a brief description of tasks you should consider in the tuning_description column, along with a SQL command you can run, where appropriate, in the tuning_command column. In records 1 and 2 above, WLA recommends that you run the Database Designer on two tables, and in record 3 recommends setting a user's password. Record 3 also provides the ALTER USER command to run because the tuning action is a SQL command.

Output in the tuning_cost column indicates the cost of running the recommended tuning command:

- **LOW**: Running the tuning command has minimal impact on resources. You can perform the tuning operation at any time, like changing the user's password in Record 3 above.

- **MEDIUM**: Running the tuning command has moderate impact on resources.

- **HIGH**: Running the tuning command has maximum impact on resources. Depending on the size of your database or table, consider running high-cost operations during off-peak load times.

Examples

The following statement tells WLA to analyze all events for the locations table:
WLA returns with a recommendation that you run the Database Designer on the table, an operation that, depending on the size of `locations`, might incur a high cost:

```sql
=> SELECT ANALYZE_WORKLOAD('locations');
```

The following statement analyzes workloads on all tables in the VMart example database since one week before today:

```sql
=> SELECT ANALYZE_WORKLOAD('', NOW() - INTERVAL '1 week');
```

WLA returns with the following results:

```
- [ RECORD 1 ]-------------------------------------------
  observation_count | 1
  first_observation_time | 2012-02-17 13:57:17.799003-04
  last_observation_time | 2011-04-22 12:05:26.856456-04
  tuning_parameter | public.locations
  tuning_description | run database designer on table public.locations
  tuning_command | ANALYZE_WORKLOAD('store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.store_orders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_ordered', 'store.storeOrders_fact.date_order...```
System Table Recommendations

You can also get tuning recommendations by querying system table TUNING_RECOMMENDATIONS, which returns tuning recommendation results from the last ANALYZE_WORKLOAD call.

```sql
=> SELECT * FROM tuning_recommendations;
```

System information that WLA uses for its recommendations is held in SQL system tables, so querying the TUNING_RECOMMENDATIONS system table does not run WLA.

See Also

Collecting Database Statistics

Understanding WLA Triggering Conditions

Workload Analyzer (WLA) monitors suspicious system activity and makes recommendations based on its observations. When you run Workload Analyzer, the utility returns the following information:

- The tuning description
- The objects on which WLA applies tuning action
- The suggested SQL command for you to run (where appropriate)

In rare circumstances, tuning recommendation WLA proposes might not resolve the underlying problem. Because you might occasionally need help understanding WLA’s recommendations, the following section lists some of the most common triggering conditions, along with the recommendations to resolve the issue and a pointer to more information, when available.

Common Problems

Problem

Internal configuration parameter is not the same across nodes.
Solution

Reset configuration parameter.

```sql
=> ALTER DATABASE mydb SET parameter = value;
```

Problem

An unused projection meets the following conditions:

- No queries on projection for more than 30 days but, projection's anchor table has been queried more than 10 times
- Projection's anchor table is not a temp or system table
- Projection's table is not small, where the number of bytes of disk storage in use by the projection (used_bytes) is more than 10M

Solution

Drop the projection (projection-name).

```sql
=> DROP PROJECTION public.T1_fact_super_P1_B1;
```

Problem

User with dbadmin/pseudosuperuser role has empty password.

Solution

Set the password for user.

```sql
=> ALTER USER (user) IDENTIFIED BY ('new_password');
```

Problem

Table with too many partition count.

Solution

Alter the table's partition expression.

```sql
=> ALTER TABLE (schema.table) PARTITION BY (new_partition_expression) REORGANIZE;
```

Problem

LGE threshold setting is lower than the default setting.

Solution

Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.
Problem
Tuple Mover's MoveOutInterval parameter is set greater than the default setting.

Solution
Decrease the MoveOutInterval configuration parameter setting.

```
=> ALTER DATABASE mydb SET MoveOutInterval = default-value;
```

For more information, see Monitoring Events and ACTIVE_EVENTS.

Problem
Tuple Mover has been disabled.

Solution
Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem
Too many ROS containers since the last mergeout operation; configuration parameters are set lower than the default.

Solution
Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem
Too many ROS containers since the last mergeout operation; the TM Mergeout service is disabled.

Solution
Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support.

Problem
Average CPU usage exceeds 95% for 20 minutes.

Solution
Check system processes or change the settings for PLANNEDCONCURRENCY and/or MAXCONCURRENCY for the resource pool. For details, see ALTER RESOURCE POOL and Built-In Pool Configuration.

Problem
Partitioned table data is not fully reorganized after repartitioning.

Solution
Reorganize data in partitioned table public.T1.

```sql
=> ALTER TABLE public.T1 REORGANIZE;
```

**Problem**
Table has multiple partition keys within the same ROS container.

**Solution**
Reorganize data in partitioned table public.T1.

```sql
=> ALTER TABLE public.T1 REORGANIZE;
```

**Problem**
Excessive swap activity; average memory usage exceeds 99% for 10 minutes.

**Solution**
Check system processes

**Problem**
A table does not have any Database Designer-designed projections.

**Solution**
Run database designer on table public.T1. For details, see Incremental Design.

**Problem**
Statistics are stale (no histogram or predicate falls outside histogram).

**Solution**
Run `ANALYZE_STATISTICS` on table columns.

```sql
=> SELECT analyze_statistics ('public.t.a');
```

See also Collecting Database Statistics.

**Problem**
Data distribution in segmented projection is skewed.

**Solution**
Re-segment projection public.t_super on high-cardinality column(s). For details, see Designing for Segmentation.

**Problem**
GROUP BY spill event.

**Solution**
Consider running an **incremental design** on query.
Managing the Database

This section describes how to manage the Vertica database. It includes the following topics:

- Connection Load Balancing
- Managing Nodes
- Adding Disk Space to a Node
- Tuple Mover Operations
- Managing the Tuple Mover
- Managing Workloads

Connection Load Balancing

Each client connection to a host in the Vertica cluster requires a small overhead in memory and processor time. If many clients connect to a single host, this overhead can begin to affect the performance of the database. You can attempt to spread the overhead of client connections by dictating that certain clients connect to specific hosts in the cluster. However, this manual balancing becomes difficult as new clients and hosts are added to your environment.

Connection load balancing helps automatically spread the overhead caused by client connections across the cluster by having hosts redirect client connections to other hosts. By redirecting connections, the overhead from client connections is spread across the cluster without having to manually assign particular hosts to individual clients. Clients can connect to a small handful of hosts, and they are naturally redirected to other hosts in the cluster.

Native Connection Load Balancing Overview

Native connection load balancing is a feature built into the Vertica Analytic Database server and client libraries as well as vsq1. Both the server and the client need to enable load balancing for it to function. If connection load balancing is enabled, a host in the database cluster can redirect a client’s attempt to it to another currently-active host in the cluster. This redirection is based on a load balancing policy. This redirection can only take place once, so a client is not bounced from one host to another.
Since native connection load balancing is incorporated into the Vertica client libraries, any client application that connects to Vertica transparently takes advantage of it simply by setting a connection parameter.

How you choose to implement connection load balancing depends on your network environment. Since native load connection balancing is easier to implement, you should use it unless your network configuration requires that clients be separated from the hosts in the Vertica database by a firewall.

For more about native connection load balancing, see About Native Connection Load Balancing.

**About Native Connection Load Balancing**

Native connection load balancing is a feature built into the Vertica server and client libraries that helps spread the CPU and memory overhead caused by client connections across the hosts in the database. It can prevent unequal distribution of client connections among hosts in the cluster.

Native connection load balancing must be enabled by the server and client. When native connection load balancing is enabled on both, the following process takes place when the client attempts to open a connection to Vertica:

1. The client connects to a host in the database cluster, with a connection parameter indicating that it is requesting a load-balanced connection.

2. The host chooses a host from the list of currently up hosts in the cluster, according to the current load balancing scheme.

3. The host tells the client which host it selected to handle the client’s connection.

4. If the host chose another host in the database to handle the client connection, the client disconnects from the initial host. Otherwise, the client jumps to step 6.

5. The client establishes a connection to the host that will handle its connection. The client sets this second connection request so that the second host does not interpret the connection as a request for load balancing.

6. The client connection proceeds as usual, (negotiating encryption if the connection has SSL enabled, and proceeding to authenticating the user ).

This process is transparent to the client application. The client driver automatically disconnects from the initial host and reconnects to the host selected for load balancing.
Requirements

- In mixed IPv4 and IPv6 environments, balancing only works for the address family for which you have configured native load balancing. For example, if you have configured load balancing using an IPv4 address, then IPv6 clients cannot use load balancing, however the IPv6 clients can still connect, but load balancing does not occur.

- The native load balancer returns an IP address for the client to use. This address must be one that the client can reach. If your nodes are on a private network, native load-balancing requires you to publish a public address in one of two ways:
  - Set the public address on each node. Vertica saves that address in the `export_address` field in the `NODES` system table.
  - Set the subnet on the database. Vertica saves that address in the `export_subnet` field in the `DATABASES` system table.

Load Balancing Schemes

The load balancing scheme controls how a host selects which host to handle a client connection. There are three available schemes:

- **NONE** (default): Disables native connection load balancing.

- **ROUNDROBIN**: Chooses the next host from a circular list of hosts in the cluster that are up—for example, in a three-node cluster, iterates over node1, node2, and node3, then wraps back to node1. Each host in the cluster maintains its own pointer to the next host in the circular list, rather than being a single cluster-wide state.

- **RANDOM**: Randomly chooses a host from among all hosts in the cluster that are up.

You set the native connection load balancing scheme using the `SET_LOAD_BALANCE_POLICY` function. See Enabling and Disabling Native Connection Load Balancing for instructions.

Driver Notes

- Native connection load balancing works with the ADO.NET driver's connection pooling. The connection the client makes to the initial host, and the final connection to the load-balanced host, use pooled connections if they are available.
If a client application uses the JDBC and ODBC driver with third-party connection pooling solutions, the initial connection is not pooled because it is not a full client connection. The final connection is pooled because it is a standard client connection.

Connection Failover

The client libraries include a failover feature that allow them to connect to backup hosts if the host specified in the connection properties is unreachable. When using native connection load balancing, this failover feature is only used for the initial connection to the database. If the host to which the client was redirected does not respond to the client's connection request, the client does not attempt to connect to a backup host and instead returns a connection error to the user.

Clients are redirected only to hosts that are known to be up. Thus, this sort of connection failure should only occur if the targeted host goes down at the same moment the client is redirected to it. For more information, see ADO.NET Connection Failover, JDBC Connection Failover, and ODBC Connection Failover in Connecting to Vertica.

Related Tasks
- Enabling and Disabling Native Connection Load Balancing
- Monitoring Native Connection Load Balancing
- Enabling Native Connection Load Balancing in ODBC
- Enabling Native Connection Load Balancing in JDBC
- Enabling Native Connection Load Balancing in ADO.NET

Reference Materials
- RESET_LOAD_BALANCE_POLICY
- SET_LOAD_BALANCE_POLICY
- JDBC Connection Properties
- Data Source Name (DSN) Connection Properties

Enabling and Disabling Native Connection Load Balancing

Only a database superuser can enable or disable native connection load balancing. To enable or disable load balancing, use the SET_LOAD_BALANCE_POLICY function to set the load balance policy. Setting the load balance policy to anything other than 'NONE' enables load balancing on the server. The following example enables native connection load balancing by setting the load balancing policy to ROUNDROBIN.
=> SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');

                    SET_LOAD_BALANCE_POLICY
-------------------
Successfully changed the client initiator load balancing policy to: roundrobin (1 row)

To disable native connection load balancing, use SET_LOAD_BALANCE_POLICY to set the policy to 'NONE':

=> SELECT SET_LOAD_BALANCE_POLICY('NONE');

                    SET_LOAD_BALANCE_POLICY
-------------------
Successfully changed the client initiator load balancing policy to: none (1 row)

Note: When a client makes a connection, the native load-balancer chooses a node and returns the value from the export_address column in the NODES table. The client then uses the export_address to connect. The node_address specifies the address to use for inter-node and spread communications. When a database is installed, the export_address and node_address are set to the same value. If you installed Vertica on a private address, then you must set the export_address to a public address for each node.

By default, client connections are not load balanced, even when connection load balancing is enabled on the server. Clients must set a connection parameter to indicates they are willing to have their connection request load balanced. See Enabling Native Connection Load Balancing in ADO.NET, Enabling Native Connection Load Balancing in JDBC and Enabling Native Connection Load Balancing in ODBC in Connecting to Vertica for more information.

Important: In mixed IPv4 and IPv6 environments, balancing only works for the address family for which you have configured load balancing. For example, if you have configured load balancing using an IPv4 address, then IPv6 clients cannot use load balancing, however the IPv6 clients can still connect, but load balancing does not occur.

Resetting the Load Balancing State

When the load balancing policy is ROUNDROBIN, each host in the Vertica cluster maintains its own state of which host it will select to handle the next client connection. You can reset this state to its initial value (usually, the host with the lowest-node id) using the RESET_LOAD_BALANCE_POLICY function:

=> SELECT RESET_LOAD_BALANCE_POLICY();

                    RESET_LOAD_BALANCE_POLICY
-------------------
Successfully reset stateful client load balance policies: "roundrobin".
Related Information

About Native Connection Load Balancing ............................................................. 1029

Related Tasks

Monitoring Native Connection Load Balancing ....................................................... 1033
Enabling Native Connection Load Balancing in ODBC ......................................... 4572
Enabling Native Connection Load Balancing in JDBC ......................................... 4634
Enabling Native Connection Load Balancing in ADO.NET .................................... 4710

Reference Materials

RESET_LOADBALANCEPOLICY .............................................................................. 2968
SET_LOADBALANCEPOLICY ............................................................................... 2969
JDBC Connection Properties .................................................................................... 4619
Data Source Name (DSN) Connection Properties ...................................................... 4551

Monitoring Native Connection Load Balancing

Query the LOAD_BALANCE_POLICY column of the V_CATALOG.DATABASES to determine the state of native connection load balancing on your server:

=> SELECT LOAD_BALANCE_POLICY FROM V_CATALOG.DATABASES;
LOAD_BALANCE_POLICY
------------------------
roundrobin
(1 row)

Determining to Which Node a Client Has Connected

A client can determine the node to which it has connected by querying the NODE_NAME column of the V_MONITOR.CURRENT_SESSION table:

=> SELECT NODE_NAME FROM V_MONITOR.CURRENT_SESSION;
NODE_NAME
----------------------
v_vmart_node0002
(1 row)
Enabling and Disabling Native Connection Load Balancing .............................................. 1031
Enabling Native Connection Load Balancing in ODBC .................................................. 4572
Enabling Native Connection Load Balancing in JDBC .................................................. 4634
Enabling Native Connection Load Balancing in ADO.NET ........................................... 4710

Reference Materials
RESET_LOAD_BALANCE_POLICY .................................................................................. 2968
SET_LOAD_BALANCE_POLICY .................................................................................... 2969
JDBC Connection Properties ......................................................................................... 4619
Data Source Name (DSN) Connection Properties .......................................................... 4551
Managing Nodes

Vertica provides the ability to add, remove, and replace nodes on a live cluster that is actively processing queries. This ability lets you scale the database without interrupting users.

In This Section

Stop Vertica on a Node

In some cases, you need to take down a node to perform maintenance tasks, or upgrade hardware.

1. Check the K-safety level of your cluster:

```sql
=> SELECT current_fault_tolerance FROM system;
  current_fault_tolerance
  ------------------------
       1
  (1 row)
```

Stopping the node might require you to temporarily reduce the K-safety level of the database. For details, see Lowering K-Safety to Enable Node Removal.

Caution: If you must reduce K-safety to 0, first back up the database.

2. Run Administration Tools, select Advanced Menu, and click OK.

3. Select Stop Vertica on Host and click OK.

4. Choose the host that you want to stop and click OK.

5. Return to the Main Menu, select View Database Cluster State, and click OK. The host you previously stopped should appear DOWN.

6. You can now perform maintenance.

See Restart Vertica on a Node for details about restarting Vertica on a node.
Restart Vertica on a Node

After stopping a node to perform maintenance, upgrade the hardware, or another similar task, you can bring the node back up. Performing this process reconnects the node with the database.

Restarting Vertica on a Node

1. Run Administration Tools. From the Main Menu select Restart Vertica on Host and click OK.

2. Select the database and click OK.

3. Select the host that you want to restart and click OK.

   Note: This process may take a few moments.

4. Return to the Main Menu, select View Database Cluster State, and click OK. The host you restarted now appears as UP, as shown.

<table>
<thead>
<tr>
<th>DB</th>
<th>Host</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>exampledb</td>
<td>ALL</td>
<td>UP</td>
</tr>
</tbody>
</table>

Setting Node Type

When you create a node, Vertica automatically sets its type to PERMANENT. This enables Vertica to use this node to store data. You can change a node's type with ALTER NODE, to one of the following:

- EPHEMERAL: A node that is in transition from one type to another—typically, from PERMANENT to either STANDBY or EXECUTE.
• **STANDBY**: A node that is reserved to replace any node when it goes down. When used as a replacement node, Vertica changes its type to PERMANENT. A standby node stores no segments or data until it is called to replace a down node. At that time, Vertica changes its type to PERMANENT. For more information, see Active Standby Nodes.

• **EXECUTE**: A node that is reserved for computation purposes only. An execute node contains no segments or data.

### Active Standby Nodes

An *active standby node* is a specialized type of Vertica node. An active standby node exists as a backup node, ready to replace a failed node. Unlike standard Vertica nodes, an active standby node does not perform computations or contain data. If a standard(permanent) node fails, an active standby node can replace the failed node, after the failed node exceeds the failover time limit. When it take the place of a failed node, the active standby node contains all of the projections and performs all of the calculations of the replaced node.

To deploy active standby nodes automatically, you must first configure the FailoverToStandbyAfter parameter. If possible, Vertica selects a standby node from the same fault group as the failed node. Otherwise, Vertica randomly selects an available active standby node.

If you are an administrator, you can manually replace a failed node using the ALTER NODE command. You can specify a particular standby node to replace a failed node, or you can allow Vertica to choose a node. As with automatic node replacement, Vertica defaults to a standby node from the same fault group as the failed node. If the fault group has no available standby nodes, Vertica selects any available active standby node.

### In This Section

### Creating an Active Standby Node

You can create active standby nodes at the same time that you create your database or at a later time.

**Note:** When you create an active standby node, be sure to add any necessary storage locations. For more information, refer to Adding Storage Locations.
Creating an Active Standby Node with a New Database

1. Create a database, including the nodes that you intend to use as active standby nodes.

2. Using vsql, connect to a node other than the node that you want to use as an active standby node.

3. Use the ALTER NODE SQL statement to convert the node from a permanent node to an active standby node. The following example shows a typical alter node command.

   ```sql
   => ALTER NODE v_mart_node5 STANDBY;
   ```

   Once you issue the ALTER NODE statement, the affected node goes down and restarts as an active standby node.

Creating an Active Standby Node with an Existing Database

When you are creating a new node with the intent of making it into an active standby node, shift the new node to ephemeral status as quickly as possible to prevent the cluster from moving data to the new node.

1. Add a new node to your database. Do not rebalance the database.

2. Using vsql, connect to a node other than the node that you want to use as an active standby node.

3. Use the ALTER NODE SQL statement to convert the node from a permanent node to an ephemeral node. For example, the following command would set v_mart_node5 to a node type of ephemeral.

   ```sql
   => ALTER NODE v_mart_node5 EPHERAL;
   ```

4. Rebalance the cluster to move any data from the ephemeral node.

5. Use the ALTER NODE SQL statement to convert the node from an ephemeral node to an active standby node. The following example shows v_mart_node5 moving from ephemeral status to standby status.

   ```sql
   => ALTER NODE v_mart_node5 STANDBY;
   ```
Replace a Node Using an Active Standby Node

Note: A node must be down for Vertica before you can replace it with an active standby node. If you attempt to replace a functioning node, Vertica displays an error message.

Replacing a Node with Automatic Failover

Vertica can automatically replace a failed node with an active standby node. In the General Parameters configure the FailoverToStandbyAfter parameter to allow automatic failover.

Manually Replacing a Node

1. Connect to the database with Administration Tools or vsq1.
2. Enter the command: ALTER NODE node-name REPLACE WITH standby-node-name;

Revert from an Active Standby Node

If an active standby node has replaced a Vertica node in your cluster, you can revert from the active standby node on a node-by-node basis or on a cluster-wide basis.

To revert from a single active standby node:

1. Connect to the database with Administration Tools or via vsq1.
2. Enter the command: ALTER NODE (node-name) RESET;.

To revert all cluster nodes from active standby:

1. Connect to the database with Administration Tools or via vsq1.
2. Enter the command: ALTER DATABASE (database-name) RESET STANDBY;.

If a down node cannot resume operation, Vertica ignores the reset request and leaves the standby node in place.
Fault Groups

Fault groups let you configure Vertica for your physical cluster layout. Sharing your cluster topology allows you to use Terrace Routing to reduce the buffer requirements of large queries and helps minimize the risk of correlated failures inherent in your environment, usually caused by shared resources.

Vertica automatically creates fault groups around control nodes (servers that run spread) in large cluster arrangements, placing nodes that share a control node in the same fault group. Automatic and user-defined fault groups do not include ephemeral nodes because such nodes hold no data.

Consider defining your own fault groups specific to your cluster's physical layout if you want to:

- Use terrace routing to reduce the buffer requirements of large queries.
- Reduce the risk of correlated failures. For example, by defining your rack layout, Vertica could tolerate a rack failure.
- Influence the placement of control nodes in the cluster.

Vertica supports complex, hierarchical fault groups of different shapes and sizes. The database platform provides a fault group script (DDL generator), SQL statements, system tables, and other monitoring tools.

See High Availability with Fault Groups for an overview of fault groups with a cluster topology example.

About the Fault Group Script

To help you define fault groups on your cluster, Vertica provides a script named fault_group_ddl_generator.py in the /opt/vertica/scripts directory. This script generates the SQL statements you need to run to create fault groups.

The fault_group_ddl_generator.py script does not create fault groups for you, but you can copy the output to a file. Then, when you run the helper script, you can use \i or vsql–f commands to pass the cluster topology to Vertica.

The fault group script takes the following arguments:

- The database name
- The fault group input file
For example:

```
$ python /opt/vertica/scripts/fault_group_ddl_generator.py VMartdb fault_grp_input.out
```

See Also

- Creating a Fault Group Input File
- Creating Fault Groups Using the Fault Group Script
- Dropping Fault Groups
- Monitoring Fault Groups
- Fault Groups

Creating a Fault Group Input File

Use a text editor to create a fault group input file for the targeted cluster.

The following example shows how you can create a fault group input file for a cluster that has 8 racks with 8 nodes on each rack—for a total of 64 nodes in the cluster.

1. On the first line of the file, list the parent (top-level) fault groups, delimited by spaces.

   ```
   rack1 rack2 rack3 rack4 rack5 rack6 rack7 rack8
   ```

2. On the subsequent lines, list the parent fault group followed by an equals sign (=). After the equals sign, list the nodes or fault groups delimited by spaces.

   ```
   <parent> = <child_1> <child_2> <child_n...>
   ```

   Such as:

   ```
   rack1 = v_vmart_node0001 v_vmart_node0002 v_vmart_node0003 v_vmart_node0004
   rack2 = v_vmart_node0005 v_vmart_node0006 v_vmart_node0007 v_vmart_node0008
   rack3 = v_vmart_node0009 v_vmart_node0010 v_vmart_node0011 v_vmart_node0012
   rack4 = v_vmart_node0013 v_vmart_node0014 v_vmart_node0015 v_vmart_node0016
   rack5 = v_vmart_node0017 v_vmart_node0018 v_vmart_node0019 v_vmart_node0020
   rack6 = v_vmart_node0021 v_vmart_node0022 v_vmart_node0023 v_vmart_node0024
   rack7 = v_vmart_node0025 v_vmart_node0026 v_vmart_node0027 v_vmart_node0028
   rack8 = v_vmart_node0029 v_vmart_node0030 v_vmart_node0031 v_vmart_node0032
   ```

After the first row of parent fault groups, the order in which you write the group descriptions does not matter. All fault groups that you define in this file must refer back to a
parent fault group. You can indicate the parent group directly or by specifying the child of a fault group that is the child of a parent fault group.

Such as:

```plaintext
rack1 rack2 rack3 rack4 rack5 rack6 rack7 rack8
rack1 = v_vmart_node0001 v_vmart_node0002 v_vmart_node0003 v_vmart_node0004
rack2 = v_vmart_node0005 v_vmart_node0006 v_vmart_node0007 v_vmart_node0008
rack3 = v_vmart_node0009 v_vmart_node0010 v_vmart_node0011 v_vmart_node0012
rack4 = v_vmart_node0013 v_vmart_node0014 v_vmart_node0015 v_vmart_node0016
rack5 = v_vmart_node0017 v_vmart_node0018 v_vmart_node0019 v_vmart_node0020
rack6 = v_vmart_node0021 v_vmart_node0022 v_vmart_node0023 v_vmart_node0024
rack7 = v_vmart_node0025 v_vmart_node0026 v_vmart_node0027 v_vmart_node0028
rack8 = v_vmart_node0029 v_vmart_node0030 v_vmart_node0031 v_vmart_node0032
```

After you create your fault group input file, you are ready to run the `fault_group_ddl_generator.py`. This script generates the DDL statements you need to create fault groups in Vertica.

If your Vertica database is co-located on a Hadoop cluster, and that cluster uses more than one rack, you can use fault groups to improve performance. See Configuring Rack Locality.

See Also

Creating Fault Groups Using the Fault Group Script

Creating Fault Groups Using the Fault Group Script

When you define fault groups, Vertica distributes data segments across the cluster. This allows the cluster to be aware of your cluster topology so it can tolerate correlated failures inherent in your environment, such as a rack failure. For an overview, see High Availability With Fault Groups.

Prerequisites for Defining Fault Groups

Defining fault groups requires careful and thorough network planning. You must have a solid understanding of your network topology.

**Important:** The user who creates fault groups must be a superuser.

Before you can define fault groups, you must:
- A fault group input file. For more information, see Creating a Fault Group Input File.

- An already-existing database

Run the Fault Group Script

1. As the database administrator, run the `fault_group_ddl_generator.py` script and include the following arguments:
   - The database name
   - The fault group input file

   For example:

   ```
   $ python /opt/vertica/scripts/fault_group_ddl_generator.py
   VMart fault_groups_VMart.out > fault_group_ddl.sql
   ```

   The preceding command writes the output of `fault_group_ddl_generator.py` to a file (`fault_group_ddl.sql`). This approach allows you to run a single SQL script instead of multiple DDL statements. Also consider saving your input file so you can modify fault groups later, such as after you expand the cluster or change the distribution of control nodes.

2. Using `vsql`, run the DDL statements in `fault_group_ddl.sql` or execute the commands in the file using `vsql`.

   ```
   => \\i fault_group_ddl.sql
   ```

3. If you have large cluster enabled, realign the control nodes, as shown. Otherwise, proceed to the next step.

   ```
   => SELECT REALIGN_CONTROL_NODES();
   ```

4. Save cluster changes to the spread configuration file.

   ```
   => SELECT RELOAD_SPREAD(true);
   ```

5. Use Administration Tools to restart the database.

6. Save changes to the cluster's data layout by calling the `REBALANCE_CLUSTER` function.

   ```
   => SELECT REBALANCE_CLUSTER();
   ```
See Also

- Cluster Management Functions
- Terrace Routing
- CREATE FAULT GROUP
- ALTER FAULT GROUP
- DROP FAULT GROUP
- ALTER DATABASE

Monitoring Fault Groups

You can monitor fault groups by querying Vertica system tables or by logging in to the Management Console (MC) interface.

Monitor Fault Groups Using System Tables

Use the following system tables to view information about fault groups and cluster vulnerabilities, such as the nodes the cluster cannot lose without the database going down:

- `V_CATALOG.FAULT_GROUPS`—View the hierarchy of all fault groups in the cluster.
- `V_CATALOG.CLUSTER_LAYOUT`—Observe the arrangement of the nodes participating in the data business and the fault groups that affect them. Ephemeral nodes do not appear in the cluster layout ring because they hold no data.

Monitoring Fault Groups Using Management Console

An MC administrator can monitor and highlight fault groups of interest by following these steps:

1. Click the running database you want to monitor and click Manage in the task bar.
2. Open the Fault Group View menu, and select the fault groups you want to view.
3. (Optional) Hide nodes that are not in the selected fault group to focus on fault groups of interest.
Nodes assigned to a fault group each have a colored bubble attached to the upper-left corner of the node icon. Each fault group has a unique color. If the number of fault groups exceeds the number of colors available, MC recycles the colors used previously.

Because Vertica supports complex, hierarchical fault groups of different shapes and sizes, MC displays multiple fault group participation as a stack of different-colored bubbles. The higher bubbles represent a lower-tiered fault group, which means that bubble is closer to the parent fault group, not the child or grandchild fault group.

For more information about fault group hierarchy, see High Availability With Fault Groups.

**Dropping Fault Groups**

When you remove a fault group from the cluster, be aware that the drop operation removes the specified fault group and its child fault groups. Vertica places all nodes under the parent of the dropped fault group. To see the current fault group hierarchy in the cluster, query the FAULT_GROUPS system table.

**Drop a Fault Group**

Use the DROP FAULT GROUP statement to remove a fault group from the cluster. The following example shows how you can drops the group2 fault group:

```sql
=> DROP FAULT GROUP group2;
DROP FAULT GROUP
```

**Drop All Fault Groups**

Use the ALTER DATABASE statement to drop all fault groups, along with any child fault groups, from the specified database cluster.

The following command drops all fault groups from the vmarltdb database.

```sql
=> ALTER DATABASE example$db DROP ALL FAULT GROUP;
ALTER DATABASE
```

**Add Nodes Back to a Fault Group**

To add a node back to a fault group, you must manually reassign it to a new or existing fault group. To do so, use the CREATE FAULT GROUP and ALTER FAULT GROUP..ADD NODE statements.
Large Cluster

To support scaling of existing clusters into large clusters and improve control message performance, Vertica delegates control message responsibilities to a subset of Vertica nodes, called control nodes. Control nodes communicate with each other. Other cluster nodes are assigned to a control node, which they use for control message communications.

Control nodes on large clusters

On clusters of 120 or more nodes, a large cluster layout is necessary and enabled by default. Vertica makes automatic control node assignments unless you use one of the following options:

<table>
<thead>
<tr>
<th>If you want to ...</th>
<th>Do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install a new cluster before you create a database</td>
<td>Run the Vertica installation script with the (--large-cluster\ &lt;integer&gt;) arguments. See the following topics for details:</td>
</tr>
<tr>
<td></td>
<td>• Installing Vertica with the Installation Script in Installing Vertica</td>
</tr>
<tr>
<td></td>
<td>• Installing a Large Cluster in this guide</td>
</tr>
<tr>
<td>• Expand an existing cluster for pre-existing databases</td>
<td>Use cluster management functions described in Defining and Realigning Control Nodes on an Existing Cluster.</td>
</tr>
</tbody>
</table>
If you want to … | Do this
---|---
• Change control nodes on an existing cluster |  
Influence the placement of control nodes in your cluster’s physical layout | Define fault groups to configure Vertica for your cluster. See Fault Groups for details.

## Control nodes on small clusters

If your cluster has fewer than 120 nodes, large cluster is neither necessary nor automatically applied. As a result, all nodes are control nodes. However, Vertica lets you define control nodes on any sized cluster. Some environments, such as cloud deployments that might have higher network latency, could benefit from a smaller number of control nodes.

For details, see Planning a Large Cluster Arrangement and Installing a Large Cluster.

## Planning a Large Cluster

In a large cluster layout of 120 nodes or more, nodes form a correlated failure group, governed by their control node—the node that runs control messaging (spread). If a control node fails, all nodes in its host group also fail.

This topic provides tips on how to plan for a large cluster arrangement. See Installing a Large Cluster and Large Cluster Best Practices for more information.

### Planning the number of control nodes

Configuring a large cluster requires careful and thorough network planning. You must have a solid understanding of your network topology before you configure the cluster.

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data K-Safety and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.
Specifying the number of control nodes

Vertica provides different tools to help you define the number of control nodes, depending on your current configuration. Consider the following scenarios, in which cluster nodes are distributed among three racks in different configurations:

<table>
<thead>
<tr>
<th>If you cluster fits this scenario ...</th>
<th>Consider this setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three control nodes. All other nodes are evenly distributed among the three racks.</td>
<td>Specify one control node per rack.</td>
</tr>
<tr>
<td>Five control nodes and three racks.</td>
<td>Specify two control nodes on two racks each and one control node on the final rack.</td>
</tr>
<tr>
<td>Four control nodes. One rack has twice as many nodes as the other racks.</td>
<td>Specify two control nodes on the larger rack and one control node each on the other two racks.</td>
</tr>
</tbody>
</table>

Installing a Large Cluster

Whether you are forming a new large cluster (adding all nodes for the first time) or expanding an existing cluster to a large cluster, Vertica provides two methods that let you specify the number of control nodes (the nodes that run control messaging). See the following sections for details:

- If you want to install a new large cluster
- If you want to expand an existing cluster

If you want to install a new large cluster

To configure Vertica for a new, large cluster, pass the `install_vertica` script the `--large-cluster <integer>` argument. Vertica selects the first `<integer>` number of hosts from the comma-separated `--hosts host_list` as control nodes and assigns all other hosts you specify in the `--hosts` argument to a control node based on a round-robin model.

Note: The number of hosts you include in the `--hosts` argument determines a large cluster layout, not the number of nodes you later include in the database. If you specify
120 or more hosts in the --hosts list, but you do not specifically enable large cluster by providing the --large-cluster argument, Vertica automatically enables large cluster and configures control nodes for you.

To help control nodes and the nodes assigned to them be configured for the highest possible fault tolerance, you must specify hosts in the --hosts host_list in a specific order. For example, if you have four sets of hosts on four racks, the first four nodes in the --hosts host_list must be one host from each rack in order to have one control node per rack. Then the list must consist of four hosts from each rack in line with the first four hosts. You'll continue to use this pattern of host listing for all targeted hosts. See Sample rack-based cluster hosts topology below for examples.

Tip: If you pass the --large-cluster argument a DEFAULT value instead of an <integer> value, Vertica calculates a number of control nodes based on the total number of nodes specified in the --hosts host_list argument. If you want a specific number of control nodes on the cluster, you must use the <integer> value.

For more information, see the following topics:

- Planning a Large Cluster Arrangement
- Installing Vertica with the Installation Script in Installing Vertica

Sample rack-based cluster hosts topology

This example shows a simple, multi-rack cluster layout, in which cluster nodes are evenly distributed across three racks. Each rack has one control node.
In the rack-based example:

- Rack-1, Rack-2, and Rack-3 are managed by a single network switch
- Host-1_1, Host-1_2, and Host-1_3 are control nodes
- All hosts on Rack-1 are assigned to control node Host-1_1
- All hosts on Rack-2 are assigned to control node Host-1_2
- All hosts on Rack-3 are assigned to control node Host-1_3

In the following `install_vertica` script fragment, note the order of the hosts in the `--hosts` list argument. The final arguments specifically enable large cluster and provide the number of control nodes (3):

```bash
... install_vertica --hosts Host-1-1,Host-1-2,Host-1-3,
    Host-2-1,Host-2-2,Host-2-3,Host-3-1,Host-3-2,Host-3-3,
    Host-4-1,Host-4-2,Host-4-3,Host-5-1,Host-5-2,Host-5-3 -rpm
    <vertica-package-name> <other-required-options>
    --large-cluster 3
```
After the installation process completes, use the Administration Tools to create a database. This operation generates a Vertica cluster with three control nodes and their respective associated hosts that reside on the same racks as the control node.

If you want to expand an existing cluster

When you add a node to an existing cluster, Vertica places the new node in an appropriate location within the cluster ring. Vertica then assigns the newly-added node to a control node, based on the cluster’s current allocations.

To give you more flexibility and control over which nodes run Spread, you can use the `SET_CONTROL_SET_SIZE(integer)` function. This function works like the installation script's `-large-cluster <integer>` option. See Defining and Realigning Control Nodes on an Existing Cluster for details.

**Important: The Vertica installation script cannot alter the database cluster.**

Defining and Realigning Control Nodes on an Existing Cluster

This topic describes how to set up or change control node assignments on an existing cluster using a series of cluster management functions. It assumes you already know how many control nodes the cluster needs for failover safety. See Planning a Large Cluster Arrangement for more information.

**Note:** If you are adding control nodes for the first time, run the Vertica installation script using the `-large-cluster <integer>` argument. See Installing Vertica with the Installation Script in Installing Vertica.

Setting up control nodes on an existing cluster makes the following changes to the cluster:

- Configures the number of nodes that run spread.
- Assigns each non-control cluster node to a control node.
- Saves the new layout to the spread configuration file.
- Redistributes data across the cluster to improve fault tolerance.

How to set up control nodes on an existing cluster

After you add, remove, or swap nodes on an existing cluster, perform the following steps. This procedure helps the cluster maintain adequate control messaging distribution for failover.
safety. For more details, see "Control node assignment/realignment" in Large Cluster Best Practices.

1. As the database administrator, log in to the Administration Tools and connect to the database.

2. Call the `SET_CONTROL_SET_SIZE(integer)` function with an integer argument that specifies the number of control nodes you want. For example, 4:

   ```sql
   => SELECT SET_CONTROL_SET_SIZE(4);
   ```

3. Call the `REALIGN_CONTROL NODES()` function without arguments:

   ```sql
   => SELECT REALIGN_CONTROL NODES();
   ```

4. Call the `RELOAD_SPREAD(true)` function to save changes to the spread configuration file:

   ```sql
   => SELECT RELOAD_SPREAD(true);
   ```

5. After the `RELOAD_SPREAD()` operation finishes, log back in to the Administration Tools, and restart the database.

6. Call the `REBALANCE_CLUSTER()` function to distribute data across the cluster:

   ```sql
   => SELECT REBALANCE_CLUSTER();
   ```

   **Important:** You must run `REBALANCE_CLUSTER` for fault tolerance to be realized.

For more details about the functions used in this procedure, see Cluster Management Functions in the SQL Reference Manual.

## Expanding the Database to a Large Cluster

If you have an existing database cluster that you want to expand to a large cluster (more than 120 nodes), follow these steps:

1. Log in to the Administration Tools as the database administrator and stop the database.

2. As root or a user with sudo privileges, open a BASH shell and run the `install_vertica` script with the `--add-hosts` argument, providing a comma-separated list of hosts you want to
add to an existing Vertica cluster. See Installing Vertica with the Installation Script in Installing Vertica.

3. Exit the shell and re-establish a vsq1 connection as the database administrator.

4. Log in to the Administration Tools and start the database.

5. Use the Administration Tools Advanced Menu > Cluster Management > Add Hosts option to add the standby hosts you created in Step 2.

6. Run `SET_CONTROL_SET_SIZE(integer)` to specify the number of control nodes you want on the cluster. See Defining and Realigning Control Nodes on an Existing Cluster.

7. Optionally create fault groups to further define the layout of the control nodes within the physical cluster. See Fault Groups.

Terrace Routing

By default, nodes in a Vertica cluster form a fully connected network (complete graph). Therefore, large Vertica clusters contain an extremely large number of connections between nodes. Each connection requires buffering. Thus, your current system can have many connections, causing queries to have extremely large data requirements on each node.

**Terrace routing** is a feature that can reduce the buffer requirements of large queries. Use terrace routing in situations where you have large queries and clusters with a large number of nodes. Without terrace routing, these situations would otherwise require excessive buffer space.

How Terrace Routing Changes Routing Behavior

In the case when data needs to be sent from each node across an entire network. In this case, Vertica uses the \( n^{th} \) node in each rack to send to all the nodes in all the racks. Terrace routing changes this behavior. Instead of sending to all nodes in all racks, the \( n^{th} \) node in each rack sends to the \( n^{th} \) nodes in all the other racks. It additional sends to all the nodes in its own rack.
Before Enabling Terrace Routing

Before you apply terrace routing to your database make sure you have read the following topics and completed the tasks described:

- Fault Groups
- Creating a Fault Group Input File
- Creating Fault Groups Using the Fault Group Script
- High Availability with Fault Groups

How to Apply Terrace Routing

Your Vertica Analytics Platform is not aware of the topology of the cluster on which your Vertica database is running. By applying fault groups to your database, you can make your Vertica Analytics Platform aware of your network's topology.

Implementing terrace routing requires use of fault groups. Terrace routing uses the fault-group definitions to represent the topology of the network of the cluster, because rack membership is a natural division for fault groups. In a terrace-routing approach, Vertica first distributes data within the rack (fault group) before forwarding it between racks. For information about creating fault groups, see Creating Fault Groups Using the Fault Group Script.
For this reason, terrace routing uses the network within the rack to reduce the buffering requirements for queries. Networks between racks can have a higher capacity so there is a performance tradeoff when a connection goes off-rack.

Calculate the TerraceRoutingFactor

TerraceRoutingFactor is a general parameter that sets the ratio of connections without terrace routing to the number of connections with terrace routing.

You can calculate the TerraceRoutingFactor value using this formula:

\[
\frac{(RackCount \times RackPopulation) - 1}{(RackCount + RackPopulation) - 2} = TerraceRoutingFactor
\]

*RackCount* is the number of racks or fault groups in your cluster, and *RackPopulation* is the number of nodes in each rack or fault group.

Enable Terrace Routing

The TerraceRoutingFactor is set to a default value of 1000.0, which is large enough to disable the parameter for even the largest clusters. Vertica recommends enabling terrace routing when your cluster contains 64 or more nodes or if your queries require excessive buffer space.

The following examples show how you can enable terrace routing for varying node and rack configurations and then disable it.

Set the TerraceRoutingFactor on a 64 node cluster with 8 racks and 8 nodes in each rack:

```sql
=> ALTER DATABASE MyDB SET TerraceRoutingFactor = 4.5;
```

Set the TerraceRoutingFactor on a 80 node cluster with 10 racks and 8 nodes in each rack in the current session:

```sql
=> ALTER SESSION SET TerraceRoutingFactor = 4.94;
```

Disable terrace routing:

```sql
=> ALTER DATABASE MyDB SET TerraceRoutingFactor = 1000;
```

Monitoring Large Clusters

Monitor large cluster traits by querying the following system tables:

- **V_CATALOG.LARGE_CLUSTER_CONFIGURATION_STATUS**—Shows the current spread hosts and the control designations in the catalog so you can see if they match.
- **V_MONITOR.CRITICAL_HOSTS**—Lists the hosts whose failure would cause the database to become unsafe and force a shutdown.

  Tip: The CRITICAL_HOSTS view is especially useful for large cluster arrangements. For non-large clusters, query the CRITICAL_NODES table.

You might also want to query the following system tables:

- **V_CATALOG.FAULT_GROUPS**—Shows fault groups and their hierarchy in the cluster.

- **V_CATALOG.CLUSTER_LAYOUT**—Shows the relative position of the actual arrangement of the nodes participating in the database cluster and the fault groups that affect them.

**Large Cluster Best Practices**

Keep the following best practices in mind when you are planning and managing a large cluster implementation.

**Planning the number of control nodes**

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data K-Safety and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.

See [Planning a Large Cluster Arrangement](#).

**Control node assignment/realignment**

After you specify the number of control nodes, you must update the control host's (spread) configuration files to reflect the catalog change. Certain cluster management functions might require that you run other functions or restart the database or both.

If, for example, you drop a control node, cluster nodes that point to it are reassigned to another control node. If that node fails, all the nodes assigned to it also fail, so you need to use the Administration Tools to restart the database. In this scenario, you'd call the REALIGN_CONTROL_NODES() and RELOAD_SPREAD(true) functions, which notify nodes of the
changes and realign fault groups. Calling RELOAD_SPREAD(true) connects an existing cluster node to a newly-assigned control node.

On the other hand, if you run REALIGN_CONTROL_NODES() multiple times in a row, the layout does not change beyond the initial setup, so you don't need to restart the database. But if you add or drop a node and then run REALIGN_CONTROL_NODES(), the function call could change many node assignments.

Here's what happens with control node assignments when you add or drop nodes, whether those nodes are control nodes or non-control nodes:

- **If you add a cluster node**—Vertica assigns a control node to the newly-added node based on the current cluster configuration. If the new node joins a fault group, it is assigned to a control node from that fault group and requires a database restart to reconnect to that control node. See Fault Groups for more information.

- **If you drop a non-control node**—Vertica quietly drops the cluster node. This operation could change the cluster and spread layout, so you must call REBALANCE_CLUSTER() after you drop a node.

- **If you drop a control node**—All nodes assigned to the control node go down. In large cluster implementations, however, the database remains up because the down nodes are not buddies with other cluster nodes.

  Dropping a control node results in \((n-1)\) control nodes. You must call REALIGN_CONTROL_NODES() to reset the cluster so it has \(n\) control nodes, which might or might not be the same number as before you dropped the control node. Remaining nodes are assigned new control nodes. In this operation, Vertica makes control node assignments based on the cluster layout. When it makes the new assignments, it respects user-defined fault groups, if any, which you can view by querying the V_CATALOG.CLUSTER_LAYOUT system table, a view that also lets you see the proposed new layout for nodes in the cluster. If you want to influence the layout of control nodes in the cluster, you should define fault groups.

For more information, see Defining and Realigning Control Nodes on an Existing Cluster and Rebalancing Data Across Nodes.

Allocate standby nodes

Have as many standby nodes available as you can, ideally on racks you are already using in the cluster. If a node suffers a non-transient failure, use the Administration Tools "Replace Host" utility to swap in a standby node.

Standby node availability is especially important for control nodes. If you are swapping a node that's a control node, all nodes assigned to the control node's host grouping will need to be
taken offline while you swap in the standby node. For details on node replacement, see \textit{Replacing Nodes}.

\textbf{Plan for cluster growth}

If you plan to expand an existing cluster to 120 or more nodes, you can configure the number of control nodes for the cluster after you add the new nodes. See \textit{Defining and Realigning Control Nodes}.

\textbf{Write custom fault groups}

When you deploy a large cluster, Vertica automatically creates fault groups around control nodes, placing nodes that share a control node into the same fault group. Alternatively, you can specify which cluster nodes should reside in a particular correlated failure group and share a control node. See \textit{High Availability With Fault Groups} in Vertica Concepts.

\textbf{Use segmented projections}

On large-cluster setups, minimize the use of unsegmented projections in favor of segmented projections. When you use segmented projections, Vertica creates buddy projections and distributes copies of segmented projections across database nodes. If a node fails, data remains available on the other cluster nodes.

\textbf{Use the Database Designer}

OpenText recommends that you use the Database Designer to create your physical schema. If you choose to design projections manually, you should segment large tables across all database nodes and replicate (unsegment) small table projections on all database nodes.

\textbf{Elastic Cluster}

You can scale your cluster up or down to meet the needs of your database. The most common case is to add nodes to your database cluster to accommodate more data and provide better query performance. However, you can scale down your cluster if you find that it is overprovisioned or if you need to divert hardware for other uses.

You scale your cluster by adding or removing nodes. Nodes can be added or removed without having to shut down or restart the database. After adding a node or before removing a node, Vertica begins a rebalancing process that moves data around the cluster to populate the new nodes or move data off of nodes about to be removed from the database. During this process,
data can be exchanged between nodes that are not being added or removed to maintain robust intelligent K-safety. If Vertica determines that the data cannot be rebalanced in a single iteration due to a lack of disk space, then the rebalance is done in multiple iterations.

To help make data rebalancing due to cluster scaling more efficient, Vertica locally segments data storage on each node so it can be easily moved to other nodes in the cluster. When a new node is added to the cluster, existing nodes in the cluster give up some of their data segments to populate the new node and exchange segments to keep the number of nodes that any one node depends upon to a minimum. This strategy keeps to a minimum the number of nodes that may become critical when a node fails (see Critical Nodes/K-Safety). When a node is being removed from the cluster, all of its storage containers are moved to other nodes in the cluster (which also relocates data segments to minimize nodes that may become critical when a node fails). This method of breaking data into portable segments is referred to as elastic cluster, since it makes enlarging or shrinking the cluster easier.

The alternative to elastic cluster is to resegment all of the data in the projection and redistribute it to all of the nodes in the database evenly any time a node is added or removed. This method requires more processing and more disk space, since it requires all of the data in all projections to essentially be dumped and reloaded.

### Elastic Cluster Scaling Factor

In a new installation, each node has a *scaling factor* that specifies the number of local segments (see Scaling Factor). Rebalance efficiently redistributes data by relocating local segments provided that, after nodes are added or removed, there are sufficient local segments in the cluster to redistribute the data evenly (determined by MAXIMUM_SKEW_PERCENT). For example, if the scaling factor = 8, and there are initially 5 nodes, then there are a total of 40 local segments cluster wide.

If you add two additional nodes (7 nodes) Vertica relocates 5 local segments on 2 nodes, and 6 such segments on 5 nodes, resulting in roughly a 16.7% skew. Rebalance chooses relocates local segments only if the resulting skew is less than the allowed threshold, as determined by MAXIMUM_SKEW_PERCENT. Otherwise, segmentation space (and hence data, if uniformly distributed over this space) is evenly distributed among the 7 nodes, and new local segment boundaries are drawn for each node, such that each node again has 8 local segments.

**Note:** By default, the scaling factor only has an effect while Vertica rebalances the database. While rebalancing, each node breaks the projection segments it contains into storage containers, which it then moves to other nodes if necessary. After rebalancing, the data is recombined into ROS containers. It is possible to have Vertica always group data into storage containers. See Local Data Segmentation for more information.
Enabling and Disabling Elastic Cluster

You enable and disable elastic cluster using functions. See the entries for the `ENABLE_ELASTIC_CLUSTER` and `DISABLE_ELASTIC_CLUSTER` functions in the SQL Reference Manual.

Query the `ELASTIC_CLUSTER` system table to determine if elastic cluster is enabled:

```sql
=> SELECT is_enabled FROM ELASTIC_CLUSTER;
 is_enabled  
 --------
 t
 (1 row)
```

Scaling Factor

To avoid an increased number of ROS containers, do not enable local segmentation and do not change the scaling factor.

Viewing Scaling Factor Settings

To view the scaling factor, query the `ELASTIC_CLUSTER` table:

```sql
=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
 scaling_factor  
 ---------------
 4
 (1 row)

=> SELECT SET_SCALING_FACTOR(6);
 SET_SCALING_FACTOR  
 -------------------
 SET
 (1 row)

=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
 scaling_factor  
 ---------------
 6
 (1 row)
```

Setting the Scaling Factor

The scaling factor determines the number of storage containers that Vertica uses to store each projection across the database during rebalancing when local segmentation is enabled. When setting the scaling factor, follow these guidelines:
The number of storage containers should be greater than or equal to the number of partitions multiplied by the number of local segments:

\[ \text{num-storage-containers} \geq (\text{num-partitions} \times \text{num-local-segments}) \]

Set the scaling factor high enough so rebalance can transfer local segments to satisfy the skew threshold, but small enough so the number of storage containers does not result in too many ROS containers, and cause ROS pushback. The maximum number of ROS containers is 1024.

Use the `SET_SCALING_FACTOR` function to change your database's scaling factor. The scaling factor can be an integer between 1 and 32.

```
=> SELECT SET_SCALING_FACTOR(12);
SET_SCALING_FACTOR
----------------------
SET (1 row)
```

Local Data Segmentation

By default, the scaling factor only has an effect when Vertica rebalances the database. During rebalancing, nodes break the projection segments they contain into storage containers which they can quickly move to other nodes.

This process is more efficient than re-segmenting the entire projection (in particular, less free disk space is required), but it still has significant overhead, since storage containers have to be separated into local segments, some of which are then transferred to other nodes. This overhead is not a problem if you rarely add or remove nodes from your database.

However, if your database is growing rapidly and is constantly busy, you may find the process of adding nodes becomes disruptive. In this case, you can enable local segmentation, which tells Vertica to always segment its data based on the scaling factor, so the data is always broken into containers that are easily moved. Having the data segmented in this way dramatically speeds up the process of adding or removing nodes, since the data is always in a state that can be quickly relocated to another node. The rebalancing process that Vertica performs after adding or removing a node just has to decide which storage containers to relocate, instead of first having to first break the data into storage containers.

Local data segmentation increases the number of storage containers stored on each node. This is not an issue unless a table contains many partitions. For example, if the table is partitioned by day and contains one or more years. If local data segmentation is enabled, then each of these table partitions is broken into multiple local storage segments, which potentially results
in a huge number of files which can lead to ROS "pushback." Consider your table partitions and the effect enabling local data segmentation may have before enabling the feature.

Enabling and Disabling Local Segmentation

To enable local segmentation, use the `ENABLE_LOCAL_SEGMENTS` function. To disable local segmentation, use the `DISABLE_LOCAL_SEGMENTATION` function:

```sql
=> SELECT ENABLE_LOCAL_SEGMENTS();
ENABLE_LOCAL_SEGMENTS
-----------------------
ENABLED
(1 row)

=> SELECT is_local_segment_enabled FROM elastic_cluster;
is_enabled
-----------
t
(1 row)

=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----------------------
DISABLED
(1 row)

=> SELECT is_local_segment_enabled FROM ELASTIC_CLUSTER;
is_enabled
-----------
f
(1 row)
```

Elastic Cluster Best Practices

The following are some best practices with regard to local segmentation.

**Note:** You should always perform a database backup before and after performing any of the operations discussed in this topic. You need to back up before changing any elastic cluster or local segmentation settings to guard against a hardware failure causing the rebalance process to leave the database in an unusable state. You should perform a full backup of the database after the rebalance procedure to avoid having to rebalance the database again if you need to restore from a backup.

When to Enable Local Data Segmentation

Local Data Segmentation can significantly speed up the process of resizing your cluster. You should enable local data segmentation if:
• your database does not contain tables with hundreds of partitions.

• the number of nodes in the database cluster is a power of two.

• you plan to expand or contract the size of your cluster.

Local segmentation can result in an excessive number of storage containers with tables that have hundreds of partitions, or in clusters with a non-power-of-two number of nodes. If your database has these two features, take care when enabling local segmentation.

Monitoring Elastic Cluster Rebalancing

Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

• The `REBALANCE_TABLE_STATUS` table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

  **Note:** If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

• `REBALANCE_PROJECTION_STATUS` can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, `separated_percent` and `transferred_percent` can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "where is_latest" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the table_id and "<unknown>" is displayed for the table_name. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.
Adding Nodes

There are many reasons for adding one or more nodes to an installation of Vertica:

- **Increase system performance.** Add additional nodes due to a high query load or load latency or increase disk space without adding storage locations to existing nodes.

  **Note:** The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

- **Make the database K-safe (K-safety=1) or increase K-safety to 2.** See Failure Recovery for details.

- **Swap a node for maintenance.** Use a spare machine to temporarily take over the activities of an existing node that needs maintenance. The node that requires maintenance is known ahead of time so that when it is temporarily removed from service, the cluster is not vulnerable to additional node failures.

- **Replace a node.** Permanently add a node to remove obsolete or malfunctioning hardware.

  **Important:** If you install Vertica on a single node without specifying the IP address or host name (or you used localhost), you cannot expand the cluster. You must reinstall Vertica and specify an IP address or host name that is not localhost/127.0.0.1.

Adding nodes consists of the following general tasks:

1. **Back up the database.**

   Vertica strongly recommends that you back up the database before you perform this significant operation because it entails creating new projections, refreshing them, and then deleting the old projections. See Backing Up and Restoring the Database for more information.

   The process of migrating the projection design to include the additional nodes could take a while; however during this time, all user activity on the database can proceed normally, using the old projections.
2. Configure the hosts you want to add to the cluster.

   See Before you Install Vertica in Installing Vertica. You will also need to edit the hosts configuration file on all of the existing nodes in the cluster to ensure they can resolve the new host.

3. Add one or more hosts to the cluster.

4. Add the hosts you added to the cluster (in step 3) to the database.

   **Note:** When you add a "host" to the database, it becomes a "node." You can add nodes to your database using either the Administration Tools or the Management Console (See Monitoring Vertica Using Management Console.)

   You can also add nodes using the admintools command line, which allows you to preserve the specific order of the nodes you add.

After you add one or more nodes to the database, Vertica automatically distributes updated configuration files to the rest of the nodes in the cluster and starts the process of rebalancing data in the cluster. See Rebalancing Data Across Nodes for details.

### Adding Hosts to a Cluster

After you have backed up the database and configured the hosts you want to add to the cluster, you can now add hosts to the cluster using the update_vertica script.

You can use MC to add standby nodes to a database, but you cannot add hosts to a cluster using MC.

### Prerequisites and Restrictions

- If you installed Vertica on a single node without specifying the IP address or hostname (you used localhost), it is not possible to expand the cluster. You must reinstall Vertica and specify an IP address or hostname.

- If your database has more than one node already, you can add a node without stopping the server. However, if you are adding a node to a single-node installation, then you must shut down both the database and spread. If you do not, the system returns an error like the following:
$ sudo /opt/vertica/sbin/update_vertica --add-hosts v_vmart_node0005 --rpm vertica_9.0.x.x86_64.RHEL6.rpm

Vertica 9.0.x Installation Tool
Starting installation tasks...
Getting system information for cluster (this may take a while)....
Spread is running on ['v_vmart_node0001']. Vertica and spread must be stopped before adding nodes to a 1 node cluster.
Use the admin tools to stop the database, if running, then use the following command to stop spread:
/etc/init.d/spread stop (as root or with sudo)
Installation completed with errors.
Installation failed.

Procedure to Add Hosts

From one of the existing cluster hosts, run the update_vertica script with a minimum of the --add-hosts host(s) parameter (where host(s) is the hostname or IP address of the system(s) that you are adding to the cluster) and the --rpm or --deb parameter:

# /opt/vertica/sbin/update_vertica --add-hosts host(s) --rpm package

**Note:** See [Installing Vertica with the Installation Script](#) for the full list of parameters. You must also provide the same options you used when originally installing the cluster.

The update_vertica script uses all the same options as install_vertica and:

- Installs the Vertica RPM on the new host.
- Performs post-installation checks, including RPM version and N-way network connectivity checks.
- Modifies spread to encompass the larger cluster.
- Configures the Administration Tools to work with the larger cluster.

**Important Tips:**

- Consider using --large-cluster with more than 50 nodes.
- A host can be specified by the hostname or IP address of the system you are adding to the cluster. However, internally Vertica stores all host addresses as IP addresses.
- Do not use include spaces in the hostname/IP address list provided with --add-hosts if you specified more than one host.
- If a package is specified with `--rpm`/`--deb`, and that package is newer than the one currently installed on the existing cluster, then, Vertica first installs the new package on the existing cluster hosts before the newly-added hosts.

- Use the same command line parameters for the database administrator username, password, and directory path you used when you installed the cluster originally. Alternatively, you can create a properties file to save the parameters during install and then re-using it on subsequent install and update operations. See Installing Vertica Silently.

- If you are installing using `sudo`, the database administrator user (dbadmin) must already exist on the hosts you are adding and must be configured with passwords and home directory paths identical to the existing hosts. Vertica sets up passwordless ssh from existing hosts to the new hosts, if needed.

- If you initially used the `--point-to-point` option to configure spread to use direct, point-to-point communication between nodes on the subnet, then use the `--point-to-point` option whenever you run `install_vertica` or `update_vertica`. Otherwise, your cluster's configuration is reverted to the default (`broadcast`), which may impact future databases.

- The maximum number of spread daemons supported in point-to-point communication and broadcast traffic is 80. It is possible to have more than 80 nodes by using large cluster mode, which does not install a spread daemon on each node.

**Examples:**

```
--add-hosts host01 --rpm
--add-hosts 192.168.233.101
--add-hosts host02,host03
```

**Adding Nodes to a Database**

Once you have added one or more hosts to the cluster, you can add them as nodes to the database.

You can add nodes to a database using these methods:

- The Management Console interface

- The Administration Tools interface
The admintools command line (to preserve the specific order of the nodes you add)

Note: When you add nodes to a database using either the MC or the Administration Tools interface (GUI), Vertica does not always preserve the order of the host and node names. You can preserve the order of the nodes you add by using the admintools command line with the db_add_node tool. An example follows, where the -s option allows you to specify the order of the nodes you are adding.

```bash
$ admintools -t db_add_node -d sampleDB -p 'password' -s 192.0.2.1,192.0.2.2,192.0.2.3
```

To Add Nodes to a Database Using MC

Only nodes in STANDBY state are eligible for addition. STANDBY nodes are nodes included in the cluster but not yet assigned to the database.

You add nodes to a database on MC's Manage page. Click the node you want to act upon, and then click Add node in the Node List.

When you add a node, the node icon in the cluster view changes color from gray (empty) to green as the node comes online. Additionally, a task list displays detailed progress of the node addition process.

To Add Nodes to a Database Using the Administration Tools:

1. Open the Administration Tools. (See Using the Administration Tools.)

2. On the Main Menu, select View Database Cluster State to verify that the database is running. If it is not, start it.

3. From the Main Menu, select Advanced Tools Menu and click OK.

4. In the Advanced Menu, select Cluster Management and click OK.

5. In the Cluster Management menu, select Add Host(s) and click OK.

6. Select the database to which you want to add one or more hosts, and then select OK.

A list of unused hosts is displayed.

7. Select the hosts you want to add to the database and click OK.

8. When prompted, click Yes to confirm that you want to add the hosts.
9. When prompted, enter the password for the database, and then select OK.

10. When prompted that the hosts were successfully added, select OK.

11. Vertica now automatically starts the rebalancing process to populate the new node with data. When prompted, enter the path to a temporary directory that the Database Designer can use to rebalance the data in the database and select OK.

12. Either press enter to accept the default K-Safety value, or enter a new higher value for the database and select OK.

13. Select whether Vertica should immediately start rebalancing the database, or whether it should create a script to rebalance the database later. You should select the option to automatically start rebalancing unless you want to delay rebalancing until a time when the database has a lower load. If you choose to automatically rebalance the database, the script is still created and saved where you can use it later.

14. Review the summary of the rebalancing process and select Proceed.

15. If you chose to automatically rebalance, the rebalance process runs. If you chose to create a script, the script is generated and saved. In either case, you are shown a success screen, and prompted to select OK to end the Add Node process.
Removing Nodes

Although less common than adding a node, permanently removing a node is useful if the host system is obsolete or over-provisioned.

Note: You cannot remove nodes if doing so leaves your cluster without the minimum number of nodes required to maintain your database's current K-safety level (3 nodes for a database with a K-safety level of 1, and 5 nodes for a K-safety level of 2). If you really wish to remove the node or nodes from the database, you first must reduce the K-safety level of your database.

This section includes:

Automatic Eviction of Unhealthy Nodes

To decrease the impact of an unhealthy node in your Vertica database, Vertica performs regular health checks. The health checks are performed on a regular schedule. The time between each interval is set by the user using the DatabaseHeartBeatInterval parameter. This parameter specifies the time intervals between internal health checks performed by each node. After a successful health check, the node sends a heartbeat. If a heartbeat is not detected by the time five intervals have elapsed, then the node is evicted from the database cluster.

The formula for the amount of time allowed before an eviction is

\[ TOT = DHBI \times 5 \]

where \( TOT \) is the total time (in seconds) allowed without a heartbeat before eviction, and \( DHBI \) is equal to the value of DatabaseHeartBeatInterval.

By default DatabaseHeartBeatInterval is set to 120, which allows five 120-second intervals to pass without a heartbeat. If you set the DatabaseHeartBeatInterval too low, it can cause evictions in cases of brief node health issues. Sometimes, such premature evictions result in lower availability and performance of the Vertica database.

See Also

DatabaseHeartbeatInterval in General Parameters
Lowering K-Safety to Enable Node Removal

A database with a K-safety level of 1 requires at least three nodes to operate, and a database with a K-Safety level 2 requires at least 5 nodes to operate. To remove a node from a cluster that is at the minimum number of nodes for its database's K-safety level, first lower the K-safety level with `MARK_DESIGN_KSAFE`.

Caution: Lowering the K-safety level of a database to 0 eliminates Vertica's fault tolerance features. If you must reduce K-safety to 0, first back up the database.

To lower the K-safety level of the database:

1. Connect to the database with Administration Tools or vsql.
2. Call the function `MARK_DESIGN_KSAFE`:

   ```sql
   SELECT MARK_DESIGN_KSAFE(n);
   ```

   where `n` is the new K-safety level for the database.

Removing Nodes From a Database

As long as there are enough nodes remaining to satisfy the K-Safety requirements, you can remove the node from a database. You cannot drop nodes that are critical for K-safety. See Lowering K-Safety to Enable Node Removal

You can remove nodes from a database using one of the following methods:

- The Management Console interface
- The Administration Tools interface

Prerequisites

Before attempting to remove a node from the database, ensure the following prerequisites are met:

- Your database must be running.
- You must back up the database.
• If necessary, lower the K-safety of your database if the cluster will not be large enough to support its current level of K-safety after you remove nodes. Remember you must have enough nodes to satisfy K-Safety requirements.

Remove Hosts From the Database with Management Console

Remove nodes with Management Console from the Manage page. You can only remove nodes that belong to the database cluster and have a DOWN state (red). Remove nodes as follows:

1. Select the node you want to remove.
2. Click Remove node in the Node List.

When you remove a node the state changes to STANDBY. You can add STANDBY nodes back to the database later, see Using the Management Console to Replace Nodes.

Remove Hosts From the Database with Administration Tools

To remove unused hosts from the database using Administration Tools:

1. Open the Administration Tools. See Using the Administration Tools for information about accessing the Administration Tools.
2. On the Main Menu, select View Database Cluster State to verify that the database is running. If the database is not running, start it.
3. From the Main Menu, select Advanced Tools Menu and select OK.
4. In the Advanced menu, select Cluster Management and select OK.
5. In the Cluster Management menu, select Remove Host(s) from Database and select OK.
6. When warned that you must redesign your database and create projections that exclude the hosts you are going to drop, select Yes.
7. Select the database from which you want to remove the hosts and select OK.
   
   A list of currently active hosts appears.
8. Select the hosts you want to remove from the database and select OK.
9. When prompted, select OK to confirm that you want to remove the hosts.
10. When informed that the hosts were successfully removed, select OK.
11. If you removed a host from a Large Cluster configuration, open a vsq1 session and run the following command:

```sql
SELECT realign_control_nodes();
```

For more details, see REALIGN_CONTROL_NODES.

12. If this host is not used by any other database in the cluster, you can remove the host from the cluster. See Removing Hosts From a Cluster.

Removing Hosts From a Cluster

If a host that you removed from the database is not used by any other database, you can remove it from the cluster using the update_mmvertica script. You can leave the database running (UP) during this operation.

You can remove hosts from a database on the MC interface, but you cannot remove those hosts from a cluster.

Prerequisites

The host must not be used by any database.

Procedure to Remove Hosts

From one of the hosts in the cluster, run update_mmvertica with the --remove-hosts switch. Provide a comma-separated list of hosts to remove from an existing Vertica cluster. You can specify a host by the host name or IP address of the system.

This example removes host01, host02, and host03 from the cluster:

```bash
# /opt/vertica/sbin/update_mmvertica --remove-hosts host01,host02,host03
```

Note: See Installing Vertica with the Installation Script for the full list of parameters.

The update_mmverticascript uses all the same options as install_mmvertica and:

- Modifies the spread to match the smaller cluster.
- Configures the Administration Tools to work with the smaller cluster.
Important Tips

- Do not include spaces in the hostname list provided with `--remove-hosts` if you specified more than one host.

- If a new RPM is specified with `--rpm`, then Vertica will first install it on the existing cluster hosts before proceeding.

- Use the same command line parameters as those used when you installed the original cluster. Specifically if you used non-default values for the database administrator username, password, or directory path, provide the same when you remove hosts; otherwise; the procedure fails. Consider creating a properties file in which you save the parameters during the installation, which you can reuse on subsequent install and update operations. See [Installing Vertica Silently](#).
Replacing Nodes

If you have a K-Safe database, you can replace nodes, as necessary, without bringing the system down. For example, you might want to replace an existing node if you:

- Need to repair an existing host system that no longer functions and restore it to the cluster
- Want to exchange an existing host system for another more powerful system

**Note:** Vertica does not support replacing a node on a K-safe=0 database. Use the procedures to `add` and `remove` nodes instead.

The process you use to replace a node depends on whether you are replacing the node with:

- A host that uses the same name and IP address
- A host that uses a different name and IP address
- An active standby node

Prerequisites

- Configure the replacement hosts for Vertica. See Before you Install Vertica in Installing Vertica.

- Read the Important Tips sections under Adding Hosts to a Cluster and Removing Hosts From a Cluster.

- Ensure that the database administrator user exists on the new host and is configured identically to the existing hosts. Vertica will setup passwordless ssh as needed.

- Ensure that directories for Catalog Path, Data Path, and any storage locations are added to the database when you create it and/or are mounted correctly on the new host and have read and write access permissions for the database administrator user. Also ensure that there is sufficient disk space.

- Follow the best practice procedure below for introducing the failed hardware back into the cluster to avoid spurious full-node rebuilds.
Best Practice for Restoring Failed Hardware

Following this procedure will prevent Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

1. Reboot the machine into runlevel 1, which is a root and console-only mode.

   Runlevel 1 prevents network connectivity and keeps Vertica from attempting to reconnect to the cluster.

2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

   Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

At this point, the network activates, and Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, Vertica will delete all files associated with that projection, which could result in data loss.

Replacing a Host Using the Same Name and IP Address

If a host of an existing Vertica database is removed you can replace it while the database is running.

Note: Remember a host in Vertica consists of the hardware and operating system on which Vertica software resides, as well as the same network configurations.

You can replace the host with a new host that has the following same characteristics as the old host:

- Name
- IP address
Operating system

The OS administrator user

Directory location

Replacing the host while your database is running prevents system downtime. Before replacing a host, backup your database. See Backing Up and Restoring the Database for more information.

Replace a host using the same characteristics as follows:

1. Run install_vertica from a functioning host using the --rpm or --deb parameter:

   ```bash
   $ /opt/vertica/sbin/install_vertica --rpm <rpm_package>
   ```

   For more information see Installing Vertica.

2. Use Administration Tools from an existing node to restart the new host. See Restart Vertica on a Node.

The node automatically joins the database and recovers its data by querying the other nodes in the database. It then transitions to an UP state.

Recovering a Failed Node Using a Node with a Different IP Address

Replacing a failed node with a host system that has a different IP address from the original consists of the following steps:

1. Back up the database.

   Vertica recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. Add the new host to the cluster. See Adding Hosts to a Cluster.

3. If Vertica is still running in the node being replaced, then use the Administration Tools to Stop Vertica on Host on the host being replaced.
4. Use the Administration Tools to replace the original host with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See Replacing Hosts.

5. Use the procedure in Distributing Configuration Files to the New Host to transfer metadata to the new host.

6. Remove the host from the cluster.

7. Use the Administration Tools to restart Vertica on the host. On the Main Menu, select Restart Vertica on Host, and click OK. See Starting the Database for more information.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying other nodes within the database.

Replacing a Functioning Node Using a Different Name and IP Address

Replacing a node with a host system that has a different IP address and host name from the original consists of the following general steps:

1. Back up the database.

   Vertica recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. Add the replacement hosts to the cluster.

   At this point, both the original host that you want to remove and the new replacement host are members of the cluster.

3. Use the Administration Tools to Stop Vertica on Host on the host being replaced.

4. Use the Administration Tools to replace the original host with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See Replacing Hosts.

5. Remove the host from the cluster.

6. Restart Vertica on the host.
Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying the other nodes within the database. It then transitions to an UP state.

**Note:** If you do not remove the original host from the cluster and you attempt to restart the database, the host is not invited to join the database because its node address does not match the new address stored in the database catalog. Therefore, it remains in the INITIALIZING state.

### Using the Administration Tools to Replace Nodes

If you are replacing a node with a host that uses a different name and IP address, use the Administration Tools to replace the original host with the new host. Alternatively, you can use the Management Console to replace a node.

Replace the Original Host with a New Host Using the Administration Tools

To replace the original host with a new host using the Administration Tools:

1. Back up the database. See [Backing Up and Restoring the Database](#).
2. From a node that is up, and is not going to be replaced, open the Administration Tools.
3. On the Main Menu, select View Database Cluster State to verify that the database is running. If it’s not running, use the Start Database command on the Main Menu to restart it.
4. On the Main Menu, select Advanced Menu.
5. In the Advanced Menu, select Stop Vertica on Host.
6. Select the host you want to replace, and then click OK to stop the node.
7. When prompted if you want to stop the host, select Yes.
8. In the Advanced Menu, select Cluster Management, and then click OK.
9. In the Cluster Management menu, select Replace Host, and then click OK.
10. Select the database that contains the host you want to replace, and then click OK.

A list of all the hosts that are currently being used displays.
11. Select the host you want to replace, and then click OK.

12. Select the host you want to use as the replacement, and then click OK.

13. When prompted, enter the password for the database, and then click OK.

14. When prompted, click Yes to confirm that you want to replace the host.

15. When prompted that the host was successfully replaced, click OK.

16. In the Main Menu, select View Database Cluster State to verify that all the hosts are running. You might need to start Vertica on the host you just replaced. Use Restart Vertica on Host.

The node enters a RECOVERING state.

Caution: If you are using a K-Safe database, keep in mind that the recovering node counts as one node down even though it might not yet contain a complete copy of the data. This means that if you have a database in which K safety=1, the current fault tolerance for your database is at a critical level. If you lose one more node, the database shuts down. Be sure that you do not stop any other nodes.

Using the Management Console to Replace Nodes

On the MC Manage page, you can quickly replace a DOWN node in the database by selecting one of the STANDBY nodes in the cluster.

A DOWN node shows up as a red node in the cluster. Click the DOWN node and the Replace node button in the Node List becomes activated, as long as there is at least one node in the cluster that is not participating in the database. The STANDBY node will be your replacement node for the node you want to retire; it will appear gray (empty) until it has been added to the database, when it turns green.

Tip: You can resize the Node List by clicking its margins and dragging to the size you want.

When you highlight a node and click Replace, MC provides a list of possible STANDBY nodes to use as a replacement. After you select the replacement node, the process begins. A node replacement could be a long-running task.

MC transitions the DOWN node to a STANDBY state, while the node you selected as the replacement will assume the identity of the original node, using the same node name, and will be started.
Assuming a successful startup, the new node will appear orange with a status of RECOVERING until the recovery procedure is complete. When the recovery process completes, the replacement node will turn green and show a state of UP.
Rebalancing Data Across Nodes

Vertica can rebalance your database when you add or remove nodes. As a superuser, you can manually trigger a rebalance with Administration Tools, SQL functions, or the Management Console.

A rebalance operation can take some time, depending on the cluster size, and the number of projections and the amount of data they contain. You should allow the process to complete uninterrupted. If you must cancel the operation, call CANCEL_REBALANCE_CLUST.

Why Rebalance?

Rebalancing is useful or even necessary after you perform one of the following operations:

- Change the size of the cluster by adding or removing nodes.
- Mark one or more nodes as ephemeral in preparation of removing them from the cluster.
- Change the scaling factor of an elastic cluster, which determines the number of storage containers used to store a projection across the database.
- Set the control node size or realign control nodes on a large cluster layout.
- Specify more than 120 nodes in your initial Vertica cluster configuration.
- Modify a fault group by adding or removing nodes.

General Rebalancing Tasks

When you rebalance a database cluster, Vertica performs the following tasks for all projections, segmented and unsegmented alike:

- Distributes data based on:
  - User-defined fault groups, if specified
  - Large cluster automatic fault groups
- Ignores node-specific distribution specifications in projection definitions. Node rebalancing always distributes data across all nodes.
When rebalancing is complete, sets the Ancient History Mark the greatest allowable epoch (now).

Vertica rebalances segmented and unsegmented projections differently, as described below.

Rebalancing Segmented Projections

For each segmented projection, Vertica performs the following tasks:

1. Copies and renames projection buddies and distributes them evenly across all nodes. The renamed projections share the same base name.
2. Refreshes the new projections.
3. Drops the original projections.

Rebalancing Unsegmented Projections

For each unsegmented projection, Vertica performs the following tasks:

If adding nodes:

- Creates projection buddies on them.
- Maps the new projections to their shared name in the database catalog.

If dropping nodes: drops the projection buddies from them.

K-safety and Rebalancing

Until rebalancing completes, Vertica operates with the existing K-safe value. After rebalancing completes, Vertica operates with the K-safe value specified during the rebalance operation. The new K-safe value must be equal to or higher than current K-safety. Vertica does not support downgrading K-safety and returns a warning if you try to reduce it from its current value. For more information, see Lowering K-Safety to Enable Node Removal.

Rebalancing Failure and Projections

If a failure occurs while rebalancing the database, you can rebalance again. If the cause of the failure has been resolved, the rebalance operation continues from where it failed. However, a
failed data rebalance can result in projections becoming out of date.

To locate any such projections, query the system table `PROJECTIONS` as follows:

```sql
=> SELECT projection_name, anchor_table_name, is_prejoin, is_up_to_date FROM projections WHERE is_up_to_date = false;
```

To remove out-of-date projections, use `DROP PROJECTION`.

**Temporary Tables**

Node rebalancing has no effect on projections of temporary tables.

**Rebalancing Data Using the Administration Tools UI**

To rebalance the data in your database:

1. Open the Administration Tools. (See Using the Administration Tools.)
2. On the Main Menu, select View Database Cluster State to verify that the database is running. If it is not, start it.
3. From the Main Menu, select Advanced Menu and click OK.
4. In the Advanced Menu, select Cluster Management and click OK.
5. In the Cluster Management menu, select Re-balance Data and click OK.
6. Select the database you want to rebalance, and then select OK.
7. Enter the directory for the Database Designer outputs (for example /tmp) and click OK.
8. Accept the proposed K-safety value or provide a new value. Valid values are 0 to 2.
9. Review the message and click Proceed to begin rebalancing data.

   The Database Designer modifies existing projections to rebalance data across all database nodes with the K-safety you provided. A script to rebalance data, which you can run manually at a later time, is also generated and resides in the path you specified; for example /tmp/extend_catalog_rebalance.sql.
Important: Rebalancing data can take some time, depending on the number of projections and the amount of data they contain. Vertica recommends that you allow the process to complete. If you must cancel the operation, use Ctrl+C.

The terminal window notifies you when the rebalancing operation is complete.

10. Press Enter to return to the Administration Tools.

Rebalancing Data Using Management Console

Vertica can rebalance the database when you add or remove nodes. If you notice data skew where one node shows more activity than another (for example, most queries processing data on a single node), you can manually rebalance the database using MC if that database is imported into the MC interface.

On the Manage page, click Rebalance in the toolbar to initiate the rebalance operation.

During a rebalance, you cannot perform any other activities on the database cluster, such as start, stop, add, or remove nodes.

Rebalancing Data Using SQL Functions

Vertica has three SQL functions for starting and stopping a cluster rebalance. You can call these functions from a script that runs during off-peak hours, rather than manually trigger a rebalance through Administration Tools.

- **REBALANCE_CLUSTER** rebalances the database cluster synchronously as a session foreground task.
- **START_REBALANCE_CLUSTER** asynchronously rebalances the database cluster as a background task.
- **CANCEL_REBALANCE_CLUSTER** stops any rebalance task that is currently in progress or is waiting to execute.

Redistributing Configuration Files to Nodes

The add and remove node processes automatically redistribute the Vertica configuration files. You rarely need to redistribute the configuration files to help resolve configuration issues.

To distribute configuration files to a host:
1. Log on to a host that contains these files and start the Administration Tools.
   See Using the Administration Tools for information about accessing the Administration Tools.

2. On the Main Menu in the Administration Tools, select Configuration Menu and click OK.

3. On the Configuration Menu, select Distribute Config Files and click OK.

4. Select Database Configuration.

5. Select the database where you want to distribute the files and click OK.
   The vertica.conf file is distributed to all the other hosts in the database. If it previously existed on a host, it is overwritten.

6. On the Configuration Menu, select Distribute Config Files and click OK.

7. Select SSL Keys.
   The certifications and keys for the host are distributed to all the other hosts in the database. If they previously existed on a host, they are overwritten.

8. On the Configuration Menu, select Distribute Config Files and click OK.
   Select AdminTools Meta-Data.
   The Administration Tools metadata is distributed to every host in the cluster.

9. Restart the database.

   **Note:** To distribute configuration files from the command line or via scripts, use the admintools option distribute_config_files:

   ```
   admintools -t distribute_config_files
   ```

## Stopping and Starting Nodes on MC

You can start and stop one or more database nodes through the Manage page by clicking a specific node to select it and then clicking the Start or Stop button in the Node List.

   **Note:** The Stop and Start buttons in the toolbar start and stop the database, not individual nodes.
On the Databases and Clusters page, you must click a database first to select it. To stop or start a node on that database, click the View button. You'll be directed to the Overview page. Click Manage in the applet panel at the bottom of the page and you'll be directed to the database node view.

The Start and Stop database buttons are always active, but the node Start and Stop buttons are active only when one or more nodes of the same status are selected; for example, all nodes are UP or DOWN.

After you click a Start or Stop button, Management Console updates the status and message icons for the nodes or databases you are starting or stopping.

Mapping New IP Addresses

There are times when existing, operational Vertica database cluster nodes need to run on different IP addresses. Cluster nodes may also need to run based on different IP protocols, for example, when changing the protocol from broadcast to point-to-point. To run Vertica in these situations, use the `re_ip` function to re-IP and map the old addresses to the new addresses:

```
$ admintools -t re_ip -f mapfile
```

The `mapfile` references the file that you must create. A map file contains the old and new IP addresses. For details see [Re-IP Addresses with a Mapping File](#).

Use this function to re-IP in one of the following situations:

- If your Vertica database cluster has the same data and control messaging address, do one of the following:
  - Re-IP all the database cluster node IP addresses.
    ```
    $ admintools -t re_ip -f mapfile
    ```
  - Re-IP only one or some of the database cluster node IP addresses.
    ```
    $ admintools -t re_ip -f mapfile
    ```

- Re-IP the Vertica database cluster from broadcast mode to point-to-point (unicast) mode:
  ```
  $ admintools -t re_ip -d db name -T
  ```

- Re-IP the Vertica database cluster from point-to-point (unicast) mode to broadcast mode:
  ```
  $ admintools -t re_ip -d db name -U
  ```
Note: For information on changing communications protocols, see the -U and -T options under install_vertica Options.

- Re-IP the control address of the database cluster. In this case the mapping file must contain the control messaging IP address and associated broadcast address.

  $ admintools -t re_ip -f mapfile

- Re-IP only one database address without changing the admintools configuration. See Mapping IP Addresses on the Database only.

Note: The database only re-ip is useful for error recovery. The node names and IP addresses must be the same as the node information in admintools.conf. You can also run SELECT * from vs_nodes order by name to display the node information.

For more information on the options used in the above commands, see Re-IP Command-Line Options.

Re-IP and the Export IP address

By default, the node IP address and the export IP address are configured with the same IP address. The export address is the IP address of the node on the network with access to other DBMS systems. Use the export address for importing and exporting data from DBMS systems. You can manually change the export address using the instructions found here.

If you change the export address and run the re-ip command, the export address remains the same.

Example

Run the following command:

```
=> SELECT node_name, node_address, export_address FROM nodes;

<table>
<thead>
<tr>
<th>node_name</th>
<th>node_address</th>
<th>export_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_VMartDB_node0001</td>
<td>192.168.100.101</td>
<td>192.168.100.101</td>
</tr>
<tr>
<td>v_VMartDB_node0002</td>
<td>192.168.100.102</td>
<td>192.168.100.101</td>
</tr>
<tr>
<td>v_VMartDB_node0003</td>
<td>192.168.100.103</td>
<td>192.168.100.101</td>
</tr>
<tr>
<td>v_VMartDB_node0004</td>
<td>192.168.100.104</td>
<td>192.168.100.101</td>
</tr>
</tbody>
</table>
```

(4 rows)

In the above example the export_address is the default. In this case, when you run the re-ip command the export_address changes to the new node_address.
If you manually change the export address as described here, you may have something like the following:

```sql
=> SELECT node_name, node_address, export_address FROM nodes;
<table>
<thead>
<tr>
<th>node_name</th>
<th>node_address</th>
<th>export_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_VMartDB_node0001</td>
<td>192.168.100.101</td>
<td>10.10.10.1</td>
</tr>
<tr>
<td>v_VMartDB_node0002</td>
<td>192.168.100.102</td>
<td>10.10.10.2</td>
</tr>
<tr>
<td>v_VMartDB_node0003</td>
<td>192.168.100.103</td>
<td>10.10.10.3</td>
</tr>
<tr>
<td>v_VMartDB_node0004</td>
<td>192.168.100.104</td>
<td>10.10.10.4</td>
</tr>
</tbody>
</table>
```

(4 rows)

In this case, when you run the re-ip command the export_address does not change.

Finding IP Addresses

IP addresses for the hosts and nodes are stored in opt/vertica/config/admintools.conf:

```
[Cluster]
hosts = 203.0.113.111, 203.0.113.112, 203.0.113.113

[Nodes]
noded0001 = 203.0.113.111//home/dbadmin,/home/dbadmin
noded0002 = 203.0.113.112//home/dbadmin,/home/dbadmin
noded0003 = 203.0.113.113//home/dbadmin,/home/dbadmin
```

You can also display a list of IP addresses with the following:

```
$ admintools -t list_allnodes
Node | Host | State | Version | DB
-----+------|-------|---------|--------
| v_vmart_node0001 | 203.0.113.111 | UP | vertica-8.1.1.20170511 | VMart
| v_vmart_node0002 | 203.0.113.112 | UP | vertica-8.1.1.20170511 | VMart
| v_vmart_node0003 | 203.0.113.113 | UP | vertica-8.1.1.20170511 | VMart
```

Tip: Run the list_allnodes tool to help identify any issues you may be having accessing Vertica. For example, if the hosts are not communicating with each other, Unavailable appears in the Version column.

Re-IP Addresses with a Mapping File

Mapping new IP addresses includes:
• Creating a mapping file that maps the old IP addresses to the new IP addresses.

• Using the mapping file to update configuration files and the database catalog.

**Note:** If you are using control messaging for communication between hosts, you may also need to update IPs with the new controlAddress and controlBroadcast IP addresses.

The embedded messages subsystem operates based on the controlAddress IP and controlBroadcast IP when you use the -U option.

Create Mapping File

Before creating a mapping file you need to know the new IP addresses. Create a mapping file as follows:

1. If you do not already have them, obtain the new IP addresses and save them in a text file. You can save the file anywhere on your system.

2. Run the following command to obtain the old IP addresses.

   ```
   $ admintools -t list_allnodes
   Node | Host | State | Version | DB
   v_vmart_node0001 | 192.0.2.254 | UP | vertica-8.1.1.20170511 | VMart
   v_vmart_node0002 | 192.0.2.255 | UP | vertica-8.1.1.20170511 | VMart
   v_vmart_node0003 | 192.0.2.256 | UP | vertica-8.1.1.20170511 | VMart
   ```

3. Copy the contents of the Host column into the same text file as the new IP addresses. The file is in the format old address, new address:

   ```
   192.0.2.254 198.51.100.255
   192.0.2.255 198.51.100.256
   192.0.2.256 198.51.100.257
   ```

You can have the following map file formats:

**Re-IP from an old IP address to a new IP address:**

oldIPaddress newIPaddress, controlAddress (optional), controlBroadcast (optional)

In this scenario, the controlAddress and controlBroadcast are optional. If you do not include them in the map file:
The controlAddress defaults to the newIPaddress.

The controlBroadcast defaults to the host of the newIPaddress’s broadcast IP address.

For example:

<table>
<thead>
<tr>
<th>IP Address 1</th>
<th>IP Address 2</th>
<th>IP Address 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.254</td>
<td>198.51.100.255</td>
<td>192.0.100.025</td>
</tr>
<tr>
<td>192.0.2.255</td>
<td>198.51.100.256</td>
<td>192.0.100.025</td>
</tr>
<tr>
<td>192.0.2.256</td>
<td>198.51.100.257</td>
<td>192.0.100.025</td>
</tr>
</tbody>
</table>

The command for performing this Re-IP process is as follows:

```
$ admintools -t re_ip -f <mapfile>
```

Re-IP from an old IP address to a new IP address and change the control messaging mode

oldIPaddress newIPaddress, controlAddress, controlBroadcast

For example:

<table>
<thead>
<tr>
<th>IP Address 1</th>
<th>IP Address 2</th>
<th>IP Address 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.254</td>
<td>198.51.100.255</td>
<td>203.0.113.255</td>
</tr>
<tr>
<td>192.0.2.255</td>
<td>198.51.100.256</td>
<td>203.0.113.255</td>
</tr>
<tr>
<td>192.0.2.256</td>
<td>198.51.100.257</td>
<td>203.0.113.255</td>
</tr>
</tbody>
</table>

Note that the map file uses comma separators after the new IP address and controlAddress.

The command for performing this re-IP process and changing the control messaging mode to point-to-point is as follows:

```
$ admintools -t re_ip -d db name -T
```

The command for performing this re-IP process and changing the control messaging mode to broadcast is as follows:

```
$ admintools -t re_ip -d db name -U
```

Re-IP the node control address on the database only (see Mapping IP Addresses on the Database only).

nodeName nodeIPaddress controlAddress, controlBroadcast

For example:

<table>
<thead>
<tr>
<th>Node Name</th>
<th>IP Address 1</th>
<th>IP Address 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>192.0.2.254</td>
<td>192.0.2.255</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>192.0.2.255</td>
<td>192.0.2.255</td>
</tr>
</tbody>
</table>

The command for performing database-only re-IP is as follows:

```bash
$ admintools -t re_ip -f <mapfile> -O -d database
```

**Re-IP the IP Addresses**

After creating the mapping file you can re-IP the new IP addresses. The re-IP process automatically backs up admintools.conf so you can recover the original settings if necessary.

1. Stop the database.

2. Run the following command to map the old IP addresses to the new IP addresses:

```bash
$ admintools -t re_ip -f mapfile
```

**Note:** This example uses the command for performing a re-IP from an old IP address to a new IP address.

A warning occurs if:

- Any of the IP addresses is incorrectly formatted
- A duplicate old or new IP address exists in the file. For example, 192.0.2.256 appears twice in the old IP set.

If the syntax is correct and mapping begins:

- Re-maps the IP addresses as listed in the mapping file.
- Prompts you to confirm the updates to the database, unless you use the -i option.
- Updates the required local configuration files with the new IP addresses.
- Distributes the updated configuration files to the hosts using the new IP addresses.

Track these steps using the following prompts:

```bash
Parsing mapfile...
New settings for Host 192.0.2.254 are:

    address: 198.51.100.255
```
New settings for Host 192.0.2.255 are:
address: 198.51.100.256

New settings for Host 192.0.2.254 are:
address: 198.51.100.257

The following databases would be affected by this tool: Vmart

Checking DB status ...
Enter "yes" to write new settings or "no" to exit > yes
Backing up local admintools.conf ...
Writing new settings to local admintools.conf ...

Writing new settings to the catalogs of database Vmart ...
The change was applied to all nodes.
Success. Change committed on a quorum of nodes.

Initiating admintools.conf distribution ...
Success. Local admintools.conf sent to all hosts in the cluster.

3. Restart the database.

Mapping IP Addresses on the Database only

You can map IP addresses for just the database. This task involves mapping the name of the nodes in the database to the new IP addresses. This is useful for error recovery because admintools.conf does not get updated. Vertica updates only spread.conf and the catalog with the changes.

You can also map IP addresses on the database only to set controlAddress and controlBroadcast on a single database. This task allows nodes on the same host to have a different data and controlAddress.

1. Stop the database.

2. Create a mapping file in the following format:

```
nodeName IPAddress, controlAddress, controlBroadcast
```

For example:

```
192.0.2.254, 203.0.113.255, 203.0.113.258 192.0.2.256, 203.0.113.257, 203.0.113.258
```

```
vertica_node001 192.0.2.254, 203.0.113.255, 203.0.113.258
vertica_node002 192.0.2.255, 203.0.113.256, 203.0.113.258
vertica_node003 192.0.2.256, 203.0.113.257, 203.0.113.258
```
3. Run the following command to map the new IP addresses:

   ```
   $ admintools -t re_ip -f <mapfile> -0 -d database
   ```

4. Restart the database.

## Re-IP Command-Line Options

The table below lists the command-line options you can use with the `re_ip` command.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h</code> or <code>--help</code></td>
<td>Displays the online help for <code>re_ip</code>.</td>
</tr>
<tr>
<td><code>-f &lt;mapfile&gt;</code> or <code>--file=&lt;mapfile&gt;</code></td>
<td>The name of the mapping text file. What this file contains depends on the type of re-IP you want to perform. See <a href="#">Mapping New IP Addresses</a>.</td>
</tr>
</tbody>
</table>
| `-0` or `--dba-only`     | Used for error recovery. Updates and replaces data on the database cluster catalog and control messaging system. If the map text file fails, Vertica automatically recreates it when you re-run the command. The format of the map text file is:  

   ```
   NodeName AssociatedNodeIPAddress, new ControlAddress, new ControlBroadcast
   ```

   NodeName and AssociatedNodeIPAddress must be the same as those in admintools.conf.

   This option updates only one database at a time so you must use the `-d` option:  

   ```
   $ admintools -t re_ip -f mapfile -0 -d database
   ```
<p>| <code>-i</code> or <code>--noprompts</code>   | Specifies that the system does not prompt for the validation of the new settings before performing the re-IP. Prompting is on by default.                 |
| <code>-T</code> or <code>--point-to-point</code> | Sets the control messaging to point-to-point (unicast) protocol. This option updates only one database at a time so you must use the <code>-d</code> option. |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>option. You do not need a mapping text file with this option. For information on setting point-to-point communication, see the -T option in install_vertica Options</td>
</tr>
<tr>
<td>-U or --broadcast</td>
<td>Sets the control messaging to broadcast protocol. This option updates only one database at a time so you must use the -d option. You do not need a mapping text file with this option. For information on setting broadcast communication, see the -U option in install_vertica Options</td>
</tr>
</tbody>
</table>
| -d <database name> or --database=<database name> | The database name. This option is required with the following re-IP options:  
  - -O  
  - -T  
  - -U |
Managing Disk Space

Vertica detects and reports low disk space conditions in the log file so you can address the issue before serious problems occur. It also detects and reports low disk space conditions via SNMP traps if enabled.

Critical disk space issues are reported sooner than other issues. For example, running out of catalog space is fatal; therefore, Vertica reports the condition earlier than less critical conditions. To avoid database corruption when the disk space falls beyond a certain threshold, Vertica begins to reject transactions that update the catalog or data.

Caution: A low disk space report indicates one or more hosts are running low on disk space or have a failing disk. It is imperative to add more disk space (or replace a failing disk) as soon as possible.

When Vertica reports a low disk space condition, use the DISK_RESOURCE_REJECTIONS system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected.

To add disk space, see Adding Disk Space to a Node. To replace a failed disk, see Replacing Failed Disks.

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

<table>
<thead>
<tr>
<th>System table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK_STORAGE</td>
<td>Monitors the amount of disk storage used by the database on each node.</td>
</tr>
<tr>
<td>COLUMN_STORAGE</td>
<td>Monitors the amount of disk storage used by each column of each projection on each node.</td>
</tr>
<tr>
<td>PROJECTION_STORAGE</td>
<td>Monitors the amount of disk storage used by each projection on each node.</td>
</tr>
</tbody>
</table>
Adding Disk Space to a Node

This procedure describes how to add disk space to a node in the Vertica cluster.

Note: If you are adding disk space to multiple nodes in the cluster, then use the following procedure for each node, one node at a time.

To add disk space to a node:

1. If you must shut down the hardware to which you are adding disk space, then first shut down Vertica on the host where disk space is being added.

2. Add the new disk to the system as required by the hardware environment. Boot the hardware if it is was shut down.

3. Partition, format, and mount the new disk, as required by the hardware environment.

4. Create a data directory path on the new volume. For example:

   ```
   mkdir -p /myNewPath/my08/host01_data2/
   ```

5. If you shut down the hardware, then restart Vertica on the host.

6. Open a database connection to Vertica and add a storage location to add the new data directory path. Specify the node in the CREATE LOCATION, otherwise Vertica assumes you are creating the storage location on all nodes.

Replacing Failed Disks

If the disk on which the data or catalog directory resides fails, causing full or partial disk loss, perform the following steps:

1. Replace the disk and recreate the data or catalog directory.
2. Distribute the configuration file (vertica.conf) to the new host. See Distributing Configuration Files to the New Host for details.
3. Restart the Vertica on the host, as described in Restart Vertica On Host.

See Catalog and Data Files for information about finding your DATABASE_HOME_DIR.

Catalog and Data Files

For the recovery process to complete successfully, it is essential that catalog and data files be in the proper directories.

In Vertica, the catalog is a set of files that contains information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster, while the data files are unique to each node. These files are installed by default in the following directories:

/DATABASE_HOME_DIR/DATABASE_NAME/v_db_node<xxx>_catalog/ /DATABASE_HOME_DIR/DATABASE_NAME/v_db_node<xxx>_catalog/

Note: DATABASE_HOME_DIR is the path, which you can see from the Administration Tools. See Using the Administration Tools in the Administrator's Guide for details on using the interface.

To view the path of your database:

1. Run the Administration Tools.
   
   $ /opt/vertica/bin/admintools

2. From the Main Menu, select Configuration Menu and click OK.

3. Select View Database and click OK.

4. Select the database you want would like to view and click OK to see the database profile.
See Understanding the Catalog Directory for an explanation of the contents of the catalog directory.

Understanding the Catalog Directory

The catalog directory stores metadata and support files for your database. Some of the files within this directory can help you troubleshoot data load or other database issues. See Catalog and Data Files for instructions on locating your database's catalog directory. By default, it is located in the database directory. For example, if you created the VMart database in the database administrator's account, the path to the catalog directory is:

```
/home/dbadmin/VMart/v_vmart_node_nnnn_catalog
```

where `node_nnnn` is the name of the node you are logged into. The name of the catalog directory is unique for each node, although most of the contents of the catalog directory are identical on each node.

The following table explains the files and directories that may appear in the catalog directory.

**Note:** Do not change or delete any of the files in the catalog directory unless asked to do so by Vertica support.

<table>
<thead>
<tr>
<th>File or Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap-catalog.log</td>
<td>A log file generated as the Vertica server initially creates the database (in which case, the log file is only created on the node used to create the database) and whenever the database is restored from a backup.</td>
</tr>
<tr>
<td>Catalog/</td>
<td>Contains catalog information about the database, such as checkpoints.</td>
</tr>
<tr>
<td>CopyErrorLogs/</td>
<td>The default location for the COPY exceptions and rejections files generated when data in a bulk load cannot be inserted into the database. See Capturing Load Rejections and Exceptions for more information.</td>
</tr>
<tr>
<td>DataCollector/</td>
<td>Log files generated by the Data Collector.</td>
</tr>
<tr>
<td>debug_log.conf</td>
<td>Debugging information configuration file. For Vertica use only.</td>
</tr>
<tr>
<td>Epoch.log</td>
<td>Used during recovery to indicate the latest epoch that contains a complete set of data.</td>
</tr>
<tr>
<td>File or Directory</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ErrorReport.txt</td>
<td>A stack trace written by Vertica if the server process exits unexpectedly.</td>
</tr>
<tr>
<td>Libraries/</td>
<td>Contains user defined library files that have been loaded into the database. See Developing User-Defined Extensions (UDxs) in Extending Vertica. Do not change or delete these libraries through the file system. Instead, use the <code>CREATE LIBRARY</code>, <code>DROP LIBRARY</code>, and <code>ALTER LIBRARY</code> statements.</td>
</tr>
<tr>
<td>Snapshots/</td>
<td>The location where backups are stored.</td>
</tr>
<tr>
<td>tmp/</td>
<td>A temporary directory used by Vertica's internal processes.</td>
</tr>
<tr>
<td>UDxLogs/</td>
<td>Log files written by user defined functions that run in fenced mode. See Fenced Mode in Extending Vertica for more information.</td>
</tr>
<tr>
<td>vertica.conf</td>
<td>The configuration file for Vertica.</td>
</tr>
<tr>
<td>vertica.log</td>
<td>The main log file generated by the Vertica server process.</td>
</tr>
<tr>
<td>vertica.pid</td>
<td>The process ID and path to the catalog directory of the Vertica server process running on this node.</td>
</tr>
</tbody>
</table>
Reclaiming Disk Space From Deleted Table Data

You can reclaim disk space from deleted table data in several ways:

- Purge deleted records.
- Rebuild the table.
- Drop table partition.

Managing Memory

glibc enhances performance by aggressively holding onto memory, even if that memory is not being used at the time by a process. The Linux kernel's Out of Memory (OOM) Killer may terminate the Vertica process if it uses too much memory.

You can avoid this by setting the MALLOCIARENA_MAX environment variable to a nonzero value, which limits the number of arenas (memory pools available to malloc()) a process can use.

If you have write privileges on ~/.bashrc, perform the following steps on all nodes:

1. Run the following command to set MALLOCIARENA_MAX:

   ```bash
   $ export MALLOCIARENA_MAX=4
   ```

2. Log out and back in.

3. Restart the Vertica database.

4. Verify that MALLOCIARENA_MAX is set:

   ```bash
   $ set | grep ARENA
   MALLOCIARENA_MAX=4
   ```

If you don't have write privileges on ~/.bashrc, you must always start the database with the VERTICAADMINTOOLS_PASSTHROUGH environment variable.
1. Always run the following command to start the database:

   ```bash
   $ export VERTICA_ADMINTOOLS_PASSTHROUGH=MALLOC_ARENA_MAX=4 admintools -t start_db -d database_name
   ```

2. Verify on each node that MALLOC_ARENA_MAX is set for the process:

   ```bash
   $ xargs --null --max-args=1 echo < /proc/$(pgrep vertica$)/environ | grep ARENA
   MALLOC_ARENA_MAX=4
   ```

## Tuple Mover Operations

The Tuple Mover manages WOS and ROS data storage. It performs two operations:

- **Moveout** moves data from WOS to ROS. During moveout operations, the Tuple Mover also enforces storage policies for the storage location.

- **Mergeout** combines small ROS containers into larger ones and purges deleted data.

The Tuple Mover automatically performs these tasks in the background, at intervals that are set by its configuration parameters.

Each of these operations occurs at different intervals across all nodes. The Tuple Mover runs independently on each node, ensuring that storage is managed appropriately even in the event of data skew.

Tuple Mover operations are automatic and transparent, and typically require no intervention. However, Vertica provides various ways to control Tuple Mover behavior. For details, see Managing the Tuple Mover.

### Mergeout

Mergeout is the Tuple Mover process that consolidates ROS containers and purges deleted records. Over time, the number of ROS containers increases enough to affect performance. It is then necessary to merge some of the ROS containers to avoid performance degradation. At that point, the Tuple Mover performs an automatic mergeout, combining two or more ROS containers into a single container. You can think of this process as a way of **defragmenting** the ROS.
Partition Mergeout

Vertica keeps data from different partitions or partition groups separate on disk. The Tuple Mover adheres to this separation policy when it consolidates ROS containers. When a partition is first created, it typically has frequent data loads and requires regular activity from the Tuple Mover. As a partition ages, it commonly transitions to a mostly read-only workload and requires much less activity.

The Tuple Mover has two different policies for managing these different partition workloads:

- **Active partition** is the partition that was most recently created. The Tuple Mover uses a STRATA mergeout policy that keeps a collection of ROS container sizes to minimize the number of times any individual tuple is subjected to mergeout. The ActivePartitionCount parameter identifies how many partitions are being actively created.

- **Inactive partitions** are those that were not most recently created. The Tuple Mover consolidates ROS containers to a minimal set while avoiding merging containers whose size exceeds MaxMrgOutROSSizeMB.

Note: If you perform a manual mergeout with the Vertica meta-function DO_TM_TASK, all partitions are consolidated into the smallest possible number of containers, regardless of the value of the ActivePartitionCount parameter.

For details on how the Tuple Mover identifies active partitions, see Active and Inactive Partitions.

Mergeout Strata Algorithm

The mergeout operation uses a strata-based algorithm to verify that each tuple is subjected to a mergeout operation a small, constant number of times, despite the process used to load the data. The mergeout operation uses this algorithm to choose which ROS containers to merge for non-partitioned tables and for active partitions in partitioned tables.

Vertica builds strata for each active partition and for projections anchored to non-partitioned tables. The number of strata, the size of each stratum, and the maximum number of ROS containers in a stratum is computed based on disk size, memory, and the number of columns in a projection.

Merging small ROS containers before merging larger ones provides the maximum benefit during the mergeout process. The algorithm begins at stratum 0 and moves upward. It checks
to see if the number of ROS containers in a stratum has reached a value equal to or greater than the maximum ROS containers allowed per stratum. The default value is 32. If the algorithm finds that a stratum is full, it marks the projections and the stratum as eligible for mergeout.

**Mergeout of Deletion Markers**

When you delete data from the database, Vertica does not remove it. Instead, it marks the data as deleted. Using many `DELETE` statements to mark a small number of rows relative to the size of a table can result in creating many small containers, the delete vectors, to hold data marked for deletion. Each delete vector container consumes resources, so a large number of such containers can impact performance, especially during recovery.

After the Tuple Mover performs a mergeout, it looks for deletion marker containers that hold few entries. If such containers exist, the Tuple Mover merges them together into a single, larger container. This process helps lower the overhead of tracking deleted data by freeing resources used by multiple, individual containers. The Tuple Mover does not purge or otherwise affect the deleted data, but consolidates delete vectors for greater efficiency.

**Note:** You can see the number and size of the containers holding the deletion marks by viewing the `V_MONITOR.DELETE_VECTORS` system table.

**Automatic and Manual Mergeout**

Vertica periodically checks ROS storage containers to determine whether delete vectors are eligible for purge, as follows:

1. Counts the number of 'aged-out' delete vectors in each container—that is, delete vectors that are as 'old' or older than the ancient history mark (AHM) epoch.

2. Calculates the percentage of aged-out delete vectors relative to the total number of records in the same ROS container.

3. If this percentage exceeds the threshold set by configuration parameter `PurgeMergeoutPercent` (by default, 20 percent), Vertica automatically performs a mergeout on the ROS container, and permanently removes all aged-out delete vectors from the ROS container.

You can also manually remove all aged-out delete vectors from ROS containers with two Vertica meta-functions:
- `DO_TM_TASK('mergeout')`

- `PURGE`

Both functions remove all aged-out delete vectors from ROS containers, regardless of how many are in a given container.

**Moveout**

Moveout operations move data from memory (WOS) into a new ROS container. A moveout *flushes* all historical data from the WOS to the ROS.

The following illustration shows the effect of a projection moveout on a single node:

**ROS Containers**

A ROS (Read Optimized Store) container is a set of rows stored in a particular group of files. ROS containers are created by operations like Moveout or COPY DIRECT. You can query the `STORAGE_CONTAINERS` system table to see ROS containers. The ROS container layout can
differ across nodes due to data variance. Segmentation can deliver more rows to one node than another. Two data loads could fit in the WOS on one node and spill on another.

**Managing the Tuple Mover**

The Tuple Mover is preconfigured to handle typical workloads. However, some situations can require you to adjust Tuple Mover behavior. You can do so in various ways:

- Configure resource pools
- Manage data loading
- Add threads
- Manage active data partitions

**Configuring Resource Pools**

The Tuple Mover draws its resources from the TM resource pool. Adding more resources (RAM) to this pool, and changing its concurrency setting, can make the Tuple Mover more effective in dealing with high load rates.

The TM resource pool concurrency setting is determined by subtracting one from the value of MAXCONCURRENCY. This calculates the number of merges that can occur simultaneously through multiple threads. As a side effect of the concurrency setting, the Tuple Mover dedicates some threads to aggressively address small ROS containers, while other threads are reserved to work only on merges of ROS containers in the lower strata.

For the TM pool, PLANNEDCONCURRENCY must be proportional to the size of the RAM, the CPU, and the storage subsystem. Depending on the storage type, if you increase PLANNEDCONCURRENCY for the Tuple Mover threads, you might create a storage I/O bottleneck. Monitor the storage subsystem; if it becomes saturated with long I/O queues, more than two I/O queues, and long latency in read and write, adjust the PLANNEDCONCURRENCY parameter to keep the storage subsystem resources below saturation level. In addition, you might need to:

- Partition storage data files
- Adjust block-size optimization on storage subsystems such as RAID 5 or RAID 10
- Identify the optimal number of disks in the RAID array
The following statement illustrates how to increase the size of the TM resource pool and set the concurrency settings for the pool:

```sql
=> ALTER RESOURCE POOL tm MEMORYSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

The WOSDATA resource pool settings also indirectly affect the Tuple Mover. In automatic mode, INSERT and COPY commands use the concurrency setting to determine whether data is small enough to store in WOS or if it should be written to ROS. Therefore, set this value to be the number of concurrent loads you expect to perform in your database. The WOSDATA resource pool also determines how much RAM the WOS can use.

```sql
=> ALTER RESOURCE POOL wosdata MAXMEMORYSIZE '4G' PLANNEDCONCURRENCY 3;
```


### Managing Data Loads

By default, Vertica automatically decides whether the data should be placed in WOS or stored directly in ROS containers based on the amount of data processed by a COPY or INSERT command. Vertica stores large loads directly to disk and stores smaller loads in memory, which it later moves to disk.

For low-latency access to data, use small loads. The automatic Tuple Mover settings are the best option for handling such smaller loads. One exception is for single-node deployments, where a system failure would cause in-memory data to be lost. In this case, you might want to force all data loads to go directly to disk.

For high load rates, you might want the Tuple Mover to check for jobs more frequently by changing the `MergeOutInterval` and `MoveOutInterval` configuration parameters. Reduce the `MoveOutInterval` if you expect the peak load rate to fill the WOS quickly. Reduce `MergeOutInterval` if you anticipate performing many DIRECT loads or inserts.

In some cases, you might wish to control how Vertica loads data into ROS or WOS. Vertica provides various options for controlling how it loads data for specific tables and DML operations. This can also affect Tuple Mover behavior. For more information, see Choosing a Load Method.

### Adding Threads

If your database is receiving a large volume of data to load or if it is performing many DIRECT loads or inserts, consider allowing the Tuple Mover to perform more operations concurrently.
by increasing the TM resource pool until it can keep up with anticipated peak loads. For example:

```sql
=> ALTER RESOURCE POOL TM MEMORYSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```


Managing Active Data Partitions

The Tuple Mover assumes that all loads and updates to a partitioned table are targeted to one or more partitions that it identifies as active. In general, the partitions with the largest partition keys—typically, the most recently created partitions—are regarded as active. As the partition ages, it commonly transitions to a mostly read-only workload and requires much less activity.

The configuration parameter `ActivePartitionCount` determines how many partitions are active for each partitioned table in the database. By default, `ActivePartitionCount` is set to 1, so the Tuple Mover assumes that only one partition—the newest one—is active. For example, if a table is partitioned by month, the Tuple Mover expects that after the start of a new month, all activity on that table is directed at the new month's partition.

If loads and updates frequently occur to multiple partitions, you can adjust the `ActivePartitionCount` parameter accordingly. For example, if your database receives data for the current month as well as updates to the prior month, set `ActivePartitionCount` to 2. For tables partitioned by non-temporal attributes, set `ActivePartitionCount` to reflect the number of partitions that are loaded simultaneously.

For more details, see `Active and Inactive Partitions`.

See Also

- Best Practices for Managing Workload Resources

Managing Workloads

Vertica's resource management scheme allows diverse, concurrent workloads to run efficiently on the database. For basic operations, Vertica pre-configures the built-in `GENERAL pool` based on RAM and machine cores. You can customize the General pool to handle specific concurrency requirements.
You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally assign each database user to use a specific resource pool, which controls memory resources used by their requests.

User-defined pools are useful if you have competing resource requirements across different classes of workloads. Example scenarios include:

- A large batch job takes up all server resources, leaving small jobs that update a web page without enough resources. This can degrade user experience.

  In this scenario, create a resource pool to handle web page requests and ensure users get resources they need. Another option is to create a limited resource pool for the batch job, so the job cannot use up all system resources.

- An application has lower priority than other applications and you want to limit the amount of memory and number of concurrent users for the low-priority application.

  In this scenario, create a resource pool with an upper limit on the query's memory and associate the pool with users of the low-priority application.

You can also use resource pools to manage resources assigned to running queries. You can assign a run-time priority to a resource pool, as well as a threshold to assign different priorities to queries with different durations. See Managing Resources At Query Run Time for more information.

For detailed syntax of creating and managing resource pools see the following topics in the SQL Reference Manual:

**Statements**

- **ALTER RESOURCE POOL** alters a resource pool.

- **ALTER USER** associates a user with the RESOURCE POOL and MEMORYCAP parameters.

- **CREATE RESOURCE POOL** creates a resource pool.

- **CREATE USER** adds a name to the list of authorized database users and specifies that user's RESOURCE POOL and MEMORYCAP parameters.

- **DROP RESOURCE POOL** drops a user-created resource pool.

- **SET SESSION MEMORYCAP** sets the limit on amount of memory that any request issued by
the session can consume.

- **SET SESSION RESOURCE POOL** associates a user session with specified resource pool.

**System Tables**

- **RESOURCE_ACQUISITIONS** provides details of resources (memory, open file handles, threads) acquired by each request for each resource pool in the system.

- **RESOURCE_POOL_DEFAULTS (systab)** lists default values for parameters in each internal and user-defined resource pool.

- **RESOURCE_POOL_STATUS** provides configuration settings of the various resource pools in the system, including internal pools.

- **RESOURCE_POOLS** displays information about the parameters the resource pool was configured with.

- **RESOURCE_QUEUES** provides information about requests pending for various resource pools.

- **RESOURCE_REJECTIONS** monitors requests for resources that are rejected by the Resource Manager.

- **RESOURCE_REJECTION_DETAILS** records an entry for each resource request that Vertica denies. This is useful for determining if there are resource space issues, as well as which users/pools encounter problems.

- **SYSTEM_RESOURCE_USAGE** provides history about system resources, such as memory, CPU, network, disk, I/O.

**Resource Manager**

On a single-user environment, the system can devote all resources to a single query, getting the most efficient execution for that one query. More likely, your environment needs to run several queries at once, which can cause tension between providing each query the maximum amount of resources (fastest run time) and serving multiple queries simultaneously with a reasonable run time.

The Vertica Resource Manager (RM) lets you resolve this tension, while ensuring that every query is eventually serviced and that true system limits are respected at all times.
For example, when the system experiences resource pressure, the Resource Manager might queue queries until the resources become available or a timeout value is reached. In addition, when you configure various RM settings, you can tune each query's target memory based on the expected number of concurrent queries running against the system.

This section discusses the detailed architecture and operation of the Resource Manager.

Resource Manager Impact on Query Execution

The Resource Manager (RM) impacts individual query execution in various ways. When a query is submitted to the database, the following series of events occur:

1. The query is parsed, optimized to determine an execution plan, and distributed to the participating nodes.

2. The Resource Manager is invoked on each node to estimate resources required to run the query and compare that with the resources currently in use. One of the following will occur:
   - If the memory required by the query alone would exceed the machine's physical memory, the query is rejected - it cannot possibly run. Outside of significantly under-provisioned nodes, this case is very unlikely.
   - If the resource requirements are not currently available, the query is queued. The query will remain on the queue until either sufficient resources are freed up and the query runs or the query times out and is rejected.
   - Otherwise the query is allowed to run.

3. The query starts running when all participating nodes allow it to run.

Note: Once the query is running, the Resource Manager further manages resource allocation using RUNTIMEPRIORITY and RUNTIMEPRIORITYTHRESHOLD parameters for the resource pool. See Managing Resources At Query Run Time for more information.

Apportioning resources for a specific query and the maximum number of queries allowed to run depends on the resource pool configuration. See Resource Pool Architecture.

On each node, no resources are reserved or held while the query is in the queue. However, multi-node queries queued on some nodes will hold resources on the other nodes. Vertica makes every effort to avoid deadlocks in this situation.
Resource Pool Architecture

The Resource Manager handles resources as one or more resource pools, which are a pre-allocated subset of the system resources with an associated queue.

Vertica is preconfigured with a set of Built-In Pools that allocate resources to different request types, where the GENERAL pool allows for a certain concurrency level based on the RAM and cores in the machines.

Modifying and Creating Resource Pools

You can configure the built-in GENERAL pool based on actual concurrency and performance requirements, as described in Built-In Pools. You can also create custom pools to handle various classes of workloads and optionally restrict user requests to your custom pools.

You create a pool using the CREATE RESOURCE POOL command.

You can create new resource pools and configure them for memory usage, concurrency, and queue priority. In addition, you can restrict a database user or user session to use a specific resource pool. Doing so allows you to control how memory, CPU, and other resources are allocated. If you have competing resource requirements across different classes of workloads, create user-defined pools. The following graphic illustrates what database operations are executed in which resource pool. Only four of the built-in pools are shown.

Defining Secondary Resource Pools

You can define secondary resource pools to which running queries can cascade if they exceed the initial pool's RUNTIMECAP.
Identifying a Secondary Pool

Defining secondary resource pools allows you to designate a place where queries that exceed the RUNTIMECAP of the pool on which they are running can execute. This way, if a query exceeds a pool's RUNTIMECAP, the query can cascade to a pool with a larger RUNTIMECAP instead of causing an error. When a query cascades to another pool, the original pool regains the memory used by that query.

Because grant privileges are not considered on secondary pools, you can use this functionality to designate secondary resource pools for user queries without giving users explicit permission to run queries on that pool.

You can also use secondary pools as a place to store long-running queries for later. Using the PRIORITY HOLD option, you can designate a secondary pool that re-queues the queries until QUEUETIMEOUT is reached or the pool's priority is changed to a non-hold value.

Query Cascade Path

Vertica routes queries to a secondary pool when the RUNTIMECAP on an initial pool is reached. Vertica then checks the secondary pool's RUNTIMECAP value. If the secondary pool's RUNTIMECAP is greater than the initial pool's value, the query executes on the secondary pool. If the secondary pool's RUNTIMECAP is less than or equal to the initial pool's value, Vertica retries the query on the next pool in the chain until it finds a pool on which the RUNTIMECAP is greater than the initial pool's value. If the secondary pool does not have sufficient resources available to execute the query at that time, SELECT queries may re-queue, re-plan, and abort on that pool. Other types of queries will fail due to insufficient resources. If no appropriate secondary pool exists for a query, the query will error out.

The following diagram demonstrates the path a query takes to execution.
Query Execution Time Allocation

After Vertica finds an appropriate pool on which to run the query, it continues to execute that query uninterrupted. The query now has the difference of the two pools' RUNTIMECAP limits in which to complete:

\[
query\ execution\ time\ allocation = rp2\ RUNTIMECAP - rp1\ RUNTIMECAP
\]

Using the CASCADE TO Parameter

As a superuser, you can identify the secondary pool by using the CASCADE TO parameter in the CREATE RESOURCE POOL or ALTER RESOURCE POOL statement. The secondary pool must already exist as a user-defined pool or the GENERAL pool. When using CASCADE TO, you cannot create a resource pool loop.

This example demonstrates a situation where the administrator wants user1's queries to start on the user_0 resource pool, but cascade to the userOverflow pool if the queries are too long.

```sql
=> CREATE RESOURCE POOL userOverflow RUNTIMECAP '5 minutes';
=> CREATE RESOURCE POOL user_0 RUNTIMECAP '1 minutes' CASCADE TO userOverflow;
=> CREATE USER "user1" RESOURCE POOL user_0;
```

In this scenario, user1 cannot start his or her queries on the userOverflow resource pool, but because grant privileges are not considered for secondary pools, user1's queries can cascade to the userOverflow pool if they exceed the user_0 pool RUNTIMECAP. Using the secondary pool frees up space in the primary pool so short queries can run.
This example shows a situation where the administrator wants long-running queries to stay queued on a secondary pool.

```sql
=> CREATE RESOURCE POOL rp2 PRIORITY HOLD;
=> CREATE RESOURCE POOL rp1 RUNTIMECAP '2 minutes' CASCADE TO rp2;
=> SET SESSION RESOURCE_POOL = rp1;
```

In this scenario, queries that run on rp1 for more than 2 minutes will queue on rp2 until QUEUETIMEOUT is reached, at which point the queries will be rejected.

### Dropping a Secondary Pool

If you try to drop a resource pool that is a secondary pool for another resource pool, Vertica returns an error. The error lists the resource pools that depend on the secondary pool you tried to drop. To drop a secondary resource pool, first set the CASCADE TO parameter to DEFAULT on the primary resource pool, and then drop the secondary pool.

For example, you can drop resource pool rp2, which is a secondary pool for rp1, as follows:

```sql
=> ALTER RESOURCE POOL rp1 CASCADE TO DEFAULT;
=> DROP RESOURCE POOL rp2;
```

### Parameter Considerations

The secondary resource pool's CPUAFFINITYSET and CPUAFFINITYMODE is applied to the query when it enters the pool.

The query adopts the secondary pool's RUNTIMEPRIORITY at different times, depending on the following circumstances:

- **If the RUNTIMEPRIORITYTHRESHOLD timer was not started when the query was running in the primary pool, the query adopts the secondary resource pools' RUNTIMEPRIORITY when it cascades.** This happens either when the RUNTIMEPRIORITYTHRESHOLD is not set for the primary pool or the RUNTIMEPRIORITY is set to HIGH for the primary pool.

- **If the RUNTIMEPRIORITYTHRESHOLD was reached in the primary pool, the query adopts the secondary resource pools' RUNTIMEPRIORITY when it cascades.**

- **If the RUNTIMEPRIORITYTHRESHOLD was not reached in the primary pool and the secondary pool has no threshold, the query adopts the new pool's RUNTIMEPRIORITY when it cascades.**

- **If the RUNTIMEPRIORITYTHRESHOLD was not reached in the primary pool and the secondary pool has a threshold set.**
If the primary pool's \texttt{RUNTIMEPRIORITYTHRESHOLD} is greater than or equal to the secondary pool's \texttt{RUNTIMEPRIORITYTHRESHOLD}, the query adopts the secondary pool's \texttt{RUNTIMEPRIORITY} after the query reaches the \texttt{RUNTIMEPRIORITYTHRESHOLD} of the primary pool.

For example:

<table>
<thead>
<tr>
<th>RUNTIMECAP of primary pool = 5 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{RUNTIMEPRIORITYTHRESHOLD} of primary pool = 8 sec</td>
</tr>
<tr>
<td>\texttt{RUNTIMEPRIORITYTHRESHOLD} of secondary pool = 7 sec</td>
</tr>
</tbody>
</table>

In this case, the query runs for 5 seconds on the primary pool and then cascades to the secondary pool. After another 3 seconds, 8 seconds total, the query adopts the \texttt{RUNTIMEPRIORITY} of the secondary pool.

If the primary pool's \texttt{RUNTIMEPRIORITYTHRESHOLD} is less than the secondary pool's \texttt{RUNTIMEPRIORITYTHRESHOLD}, the query adopts the secondary pool's \texttt{RUNTIMEPRIORITY} after the query reaches the \texttt{RUNTIMEPRIORITYTHRESHOLD} of the secondary pool.

For example,

<table>
<thead>
<tr>
<th>RUNTIMECAP of primary pool = 5 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{RUNTIMEPRIORITYTHRESHOLD} of primary pool = 8 sec</td>
</tr>
<tr>
<td>\texttt{RUNTIMEPRIORITYTHRESHOLD} of secondary pool = 12 sec</td>
</tr>
</tbody>
</table>

In this case, the query runs for 5 seconds on the primary pool and then cascades to the secondary pool. After another 7 seconds, 12 seconds total, the query adopts the \texttt{RUNTIMEPRIORITY} of the secondary pool.

### Using Queries to Monitor Resource Pool Size and Usage

You can use the Linux \texttt{top} command to determine the overall CPU usage and I/O wait time across the system. However, because of file system caching, the resident memory size indicated by \texttt{top} is not a good indicator of actual memory use or available reserve.

Instead, Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests, and the state of the queues.

The \texttt{RESOURCE_POOLS} table lets you view various resource pools defined in the system (both internal and user-defined), and the \texttt{RESOURCE_POOL_STATUS} table lets you view the current state of the resource pools.
Examples

This example shows how to find the various resource pools defined in the system.

=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;

<table>
<thead>
<tr>
<th>name</th>
<th>memorysize</th>
<th>maxmemorysize</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td></td>
<td>Special: 95%</td>
</tr>
<tr>
<td>sysquery</td>
<td>64M</td>
<td></td>
</tr>
<tr>
<td>sysdata</td>
<td>100M</td>
<td>10%</td>
</tr>
<tr>
<td>wosdata</td>
<td>0%</td>
<td>25%</td>
</tr>
<tr>
<td>tm</td>
<td>200M</td>
<td></td>
</tr>
<tr>
<td>refresh</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>recovery</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>dbd</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>jvm</td>
<td>0%</td>
<td>10%</td>
</tr>
</tbody>
</table>

(9 rows)

Viewing Only User-Defined Resource Pools

To see only the user-defined resource pools, you can limit your query to return records where IS_INTERNAL is false.

Note: The user-defined pools shown in these examples also appear in subsequent sections related to Workload Management.

This example shows how to find information on user-defined resource pools:

=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';

<table>
<thead>
<tr>
<th>name</th>
<th>memorysize</th>
<th>maxmemorysize</th>
<th>priority</th>
<th>maxconcurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_pool</td>
<td>0%</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ceo_pool</td>
<td>250M</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ad_hoc_pool</td>
<td>200M</td>
<td>200M</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>billing_pool</td>
<td>0%</td>
<td>me</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>web_pool</td>
<td>25M</td>
<td>me</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>batch_pool</td>
<td>150M</td>
<td>150M</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>dept1_pool</td>
<td>0%</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>dept2_pool</td>
<td>0%</td>
<td></td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

(8 rows)
Viewing the Status of All Resource Pools

The following example shows how to access the `V_MONITOR.RESOURCE_POOL_STATUS` table to return the current state of all resource pools on node0001:

```sql
=>
x
Expanded display is on
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
    general_memory_borrowed_kb,running_query_count
    FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike 'node0001';

    pool_name | general
  memory_size_kb     | 2983177
  memory_size_actual_kb | 2983177
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | sydata
  memory_size_kb     | 65536
  memory_size_actual_kb | 65536
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | sysquery
  memory_size_kb     | 102400
  memory_size_actual_kb | 102400
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | wosdata
  memory_size_kb     | 0
  memory_size_actual_kb | 0
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | tm
  memory_size_kb     | 204800
  memory_size_actual_kb | 204800
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | refresh
  memory_size_kb     | 0
  memory_size_actual_kb | 0
  memory_inuse_kb     | 0
  general_memory_borrowed_kb | 0
  running_query_count | 0

    pool_name | recovery
  memory_size_kb     | 0
```
Viewing Query Resource Acquisitions

The following example displays all resources granted to the queries that are currently running. The information shown is stored in the `V_MONITOR.RESOURCE_ACQUISITIONS` table. You can see that the query execution used 708504 KB of memory from the GENERAL pool.

```sql
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
    queue_entry_timestamp, acquisition_timestamp
FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE 'node0001';
```

You can determine how long a query waits in the queue before it can run. To do so, you obtain the difference between the `acquisition_timestamp` and `queue_entry_timestamp` using a query as this example shows:
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
    (acquisition_timestamp - queue_entry_timestamp) AS 'queue wait'
FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

- [ RECORD 1 ]----------------------------------------------
pool_name | sysquery
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
queue wait | 00:00:00.000005

- [ RECORD 2 ]----------------------------------------------
pool_name | sysquery
queue_entry_timestamp | 2013-12-05 07:07:14.714412-05
acquisition_timestamp | 2013-12-05 07:07:14.714417-05
queue wait | 00:00:00.000005

- [ RECORD 3 ]----------------------------------------------
pool_name | sysquery
queue_entry_timestamp | 2013-12-05 07:09:57.238521-05
acquisition_timestamp | 2013-12-05 07:09:57.281708-05
queue wait | 00:00:00.043187

- [ RECORD 4 ]----------------------------------------------
...

See Also

- See the SQL Reference Manual for detailed descriptions of the monitoring tables.
- See Monitoring Resource Pools for descriptions of other ways to monitor resource usage.

User Profiles

User profiles are attributes associated with a user that control that user's access to several system resources. These resources include:

- Resource pool to which a user is assigned (RESOURCE POOL)
- Maximum amount of memory a user's session can use (MEMORYCAP)
- Maximum amount of temporary file storage a user's session can use (TEMPSPACECAP)
- Maximum amount of time a user's query can run (RUNTIMECAP)

You can set these attributes with the CREATE USER statement and modify the attributes later with ALTER USER.

Two strategies limit a user's access to resources: Setting attributes on the user directly to control resource use, or assigning the user to a resource pool. The first method lets you fine
tune individual users, while the second makes it easier to group many users together and set their collective resource usage.

The following examples illustrate how to set a user's resource pool attributes. For additional examples, see the scenarios described in Using User-Defined Pools and User-Profiles for Workload Management.

Example

Set the user's RESOURCE POOL attribute to assign the user to a resource pool. To create a user named user1 who has access to the resource pool my_pool, use the command:

```sql
=> CREATE USER user1 RESOURCE POOL my_pool;
```

To limit the amount of memory for a user without designating a pool, set the user's MEMORYCAP to either a particular unit or a percentage of the total memory available. For example, to create a user named user2 whose sessions are limited to using 200 MBs memory each, use the command:

```sql
=> CREATE USER user2 MEMORYCAP '200M';
```

To limit the time a user's queries are allowed to run, set the RUNTIMECAP attribute. To prevent queries for user2 from running more than five minutes, use this command:

```sql
=> ALTER USER user2 RUNTIMECAP '5 minutes';
```

To limit the amount of temporary disk space that the user's sessions can use, set the TEMPSPACECAP to either a particular size or a percentage of temporary disk space available. For example, the next statement creates user3, and limits her to using 1 GB of temporary space:

```sql
=> CREATE USER user3 TEMPSPACECAP '1G';
```

You can combine different attributes into a single command. For example, to limit the MEMORYCAP and RUNTIMECAP for user3, include both attributes in an ALTER USER statement:

```sql
=> ALTER USER user3 MEMORYCAP '750M' RUNTIMECAP '10 minutes';
```

Expanded display is on.

```sql
SELECT *
FROM USERS;
```

<table>
<thead>
<tr>
<th>user_id</th>
<th>45035996273704962</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>release</td>
</tr>
<tr>
<td>is_super_user</td>
<td>t</td>
</tr>
<tr>
<td>resource_pool</td>
<td>general</td>
</tr>
<tr>
<td>memory_cap_kb</td>
<td>unlimited</td>
</tr>
<tr>
<td>temp_space_cap_kb</td>
<td>unlimited</td>
</tr>
</tbody>
</table>

Expanded display is off.
See Also

- ALTER USER
- CREATE USER

Target Memory Determination for Queries in Concurrent Environments

The resource pool parameters MEMORYSIZE, MAXMEMORYSIZE, and PLANNEDCONCURRENCY allow you to tune the target memory allocated to queries.

Note: For details about these parameters, see CREATE RESOURCE POOL in the SQL Reference Manual.

What is the Query Budget?

Each resource pool has a query budget, which is the target memory for queries executed on the associated pool. Vertica stores this value in the query_budget_kb column in the V_MONITOR.RESOURCE_POOL_STATUS system table.
Computing the Query Budget

The formula for computing a pool's query budget is as follows:

**GENERAL pool:**

- Query budget = Queuing threshold of the GENERAL pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to 0 and MAXMEMORYSIZE is NOT set:

- Query budget = Queuing threshold of the GENERAL pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to 0 and MAXMEMORYSIZE is set to a value other than the default:

- Query budget = Queuing threshold of the pool / PLANNEDCONCURRENCY

If MEMORYSIZE is set to a value other than the default:

- Query budget = MEMORYSIZE / PLANNEDCONCURRENCY of the pool

The consequence of this is by carefully tuning the MEMORYSIZE and PLANNEDCONCURRENCY parameters, it is possible to restrict the amount of memory used by a query to a desired size. For a detailed example of query budget calculations, see Do You Need to Put Your Query on a Budget? in the Vertica User Community.

**Note:** Vertica calculates the queuing threshold for a pool is 95% of a pool's MAXMEMORYSIZE.

Tuning the Query Budget

Normally, query budgets do not require any specific tuning. However, if you reduce the MAXMEMORYSIZE because you need memory for other purposes, be aware that you are also reducing the query budget. Reducing the query budget negatively impacts the query performance, particularly if the queries are complex.

To maintain the original query budget for the resource pool, if you reduce MAXMEMORYSIZE, be sure to also reduce the value of PLANNEDCONCURRENCY.

Parameter Values

The RESOURCE_POOL_STATUS system table contains the values for parameters for all Vertica resource pools.
Managing Resources At Query Run Time

The Resource Manager estimates the resources required for queries to run, and then determines when to run queries and when to queue them.

The Resource Manager also lets you manage resources that are assigned to queries that are already running using either of these methods:

- **Setting Runtime Priority for the Resource Pool**—Use resource pool parameters to set the run time priority for queries running within the resource pool.
- **Changing Runtime Priority of a Running Query**—Manually change the run time priority of a running query.

Setting Runtime Priority for the Resource Pool

For each resource pool, you can manage resources that are assigned to queries that are already running. You assign each resource pool a runtime priority of HIGH, MEDIUM, or LOW. These settings determine the amount of runtime resources (such as CPU and I/O bandwidth) assigned to queries in the resource pool when they run. Queries in a resource pool with a HIGH priority are assigned greater runtime resources than those in resource pools with MEDIUM or LOW runtime priorities.

Prioritizing Queries Within a Resource Pool

While runtime priority helps to manage resources for the resource pool, there may be instances where you want some flexibility within a resource pool. For instance, you may want to ensure that very short queries run at a high priority, while also ensuring that all other queries run at a medium or low priority.

The Resource Manager allows you this flexibility by letting you set a **runtime priority threshold** for the resource pool. With this threshold, you specify a time limit (in seconds) by which a query must finish before it is assigned the runtime priority of the resource pool. All queries begin running with a HIGH priority; once a query's duration exceeds the time limit specified in the runtime priority threshold, it is assigned the runtime priority of the resource pool.
Setting Runtime Priority and Runtime Priority Threshold

You specify runtime priority and runtime priority threshold by setting two resource pool parameters with `CREATE RESOURCE POOL` or `ALTER RESOURCE POOL`:

- `RUNTIMEPRIORITY`
- `RUNTIMEPRIORITYTHRESHOLD`

Changing Runtime Priority of a Running Query

`CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY` lets you to change a query's runtime priority. You can change the runtime priority of a query that is already executing.

This function takes two arguments:

- The query's transaction ID, obtained from the system table `SESSIONS`
- The desired priority, one of the following string values: `HIGH`, `MEDIUM`, or `LOW`

Restrictions

Superusers can change the runtime priority of any query to any priority level. The following restrictions apply to other users:

- They can only change the runtime priority of their own queries.
- They cannot raise the runtime priority of a query to a level higher than that of the resource pools.

Procedure

Changing a query's runtime priority is a two-step procedure:

1. Get the query's transaction ID by querying the system table `SESSIONS`. For example, the following statement returns information about all running queries:

   ```sql
   => SELECT transaction_id, runtime_priority, transaction_description FROM SESSIONS;
   ```

2. Run `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY`, specifying the query's
transaction ID and desired runtime priority:

```sql
SELECT CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY(45035996273705748, 'low')
```

**Manually Moving Queries to Different Resource Pools**

If you are the database administrator, you can move queries to another resource pool mid-execution using the `MOVE_STATEMENT_TORESOURCE_POOL` meta-function.

You might want to use this feature if a single query is using a large amount of resources, preventing smaller queries from executing.

**What Happens When a Query Moves to a Different Resource Pool**

When a query is moved from one resource pool to another, it continues executing, provided the target pool has enough resources to accommodate the incoming query. If sufficient resources cannot be assigned in the target pool on at least one node, Vertica cancels the query and attempts to re-plan the query. If Vertica cannot re-plan the query, the query is canceled indefinitely.

When you successfully move a query to a target resource pool, its resources will be accounted for by the target pool and released on the first pool.

If you move a query to a resource pool with PRIORITY HOLD, Vertica cancels the query and queues it on the target pool. This cancellation remains in effect until you change the PRIORITY or move the query to another pool without PRIORITY HOLD. You can use this option if you want to store long-running queries for later use.

You can view the `RESOURCE_ACQUISITIONS` or `RESOURCE_POOL_STATUS` system tables to determine if the target pool can accommodate the query you want to move. Be aware that the system tables may change between the time you query the tables and the time you invoke the `MOVE_STATEMENT_TORESOURCE_POOL` meta-function.

When a query successfully moves from one resource pool to another mid-execution, it executes until the greater of the existing and new RUNTIMECAP is reached. For example, if the RUNTIMECAP on the initial pool is greater than that on the target pool, the query can execute until the initial RUNTIMECAP is reached.

When a query successfully moves from one resource pool to another mid-execution the CPU affinity will change.
Using the MOVE_STATEMENT_TO_RESOURCE_POOL Function

To manually move a query from its current resource pool to another resource pool, use the MOVE_STATEMENT_TORESOURCE_POOL meta-function. Provide the session id, transaction id, statement id, and target resource pool name, as shown:

```sql
=> SELECT MOVE_STATEMENT_TO_RESOURCE_POOL ('v_vmart_node0001.example.31427:0x82fbm', 45035996273711993, 1, 'my_target_pool');
```

See Also:

- Defining Secondary Resource Pools
- MOVE_STATEMENT_TO_RESOURCE_POOL
- RESOURCE_POOL_MOVE

Restoring Resource Manager Defaults

The system table V_CATALOG.RESOURCE_POOL_DEFAULTS stores default values for all parameters for all built-in and user-defined resource pools.

If you have changed the value of any parameter in any of your resource pools and want to restore it to its default, you can simply alter the table and set the parameter to DEFAULT. For example, the following statement sets the RUNTIMEPRIORITY for the resource pool sysquery back to its default value:

```sql
VMart=> ALTER RESOURCE POOL sysquery RUNTIMEPRIORITY DEFAULT;
```

See Also

- RESOURCE_POOL_DEFAULTS
Best Practices for Managing Workload Resources

This section provides general guidelines and best practices on how to set up and tune resource pools for various common scenarios.

Note: The exact settings for resource pool parameters are heavily dependent on your query mix, data size, hardware configuration, and concurrency requirements. Vertica recommends performing your own experiments to determine the optimal configuration for your system.

Basic Principles for Scalability and Concurrency Tuning

A Vertica database runs on a cluster of commodity hardware. All loads and queries running against the database take up system resources, such as CPU, memory, disk I/O bandwidth, file handles, and so forth. The performance (run time) of a given query depends on how much resource it has been allocated.

When running more than one query concurrently on the system, both queries are sharing the resources; therefore, each query could take longer to run than if it was running by itself. In an efficient and scalable system, if a query takes up all the resources on the machine and runs in X time, then running two such queries would double the run time of each query to 2X. If the query runs in > 2X, the system is not linearly scalable, and if the query runs in < 2X then the single query was wasteful in its use of resources. Note that the above is true as long as the query obtains the minimum resources necessary for it to run and is limited by CPU cycles. Instead, if the system becomes bottlenecked so the query does not get enough of a particular resource to run, then the system has reached a limit. In order to increase concurrency in such cases, the system must be expanded by adding more of that resource.

In practice, Vertica should achieve near linear scalability in run times, with increasing concurrency, until a system resource limit is reached. When adequate concurrency is reached without hitting bottlenecks, then the system can be considered as ideally sized for the workload.

Note: Typically Vertica queries on segmented tables run on multiple (likely all) nodes of the cluster. Adding more nodes generally improves the run time of the query almost linearly.
Setting a Runtime Limit for Queries

You can set a limit for the amount of time a query is allowed to run. You can set this limit at three levels, listed in descending order of precedence:

1. The resource pool to which the user is assigned.
2. User profile with RUNTIMECAP configured by CREATE USER/ALTER USER
3. Session queries, set by SET SESSION RUNTIMECAP

In all cases, you set the runtime limit with an interval value that does not exceed one year. When you set runtime limit at multiple levels, Vertica always uses the shortest value. If a runtime limit is set for a non-superuser, that user cannot set any session to a longer runtime limit. Superusers can set the runtime limit for other users and for their own sessions, to any value up to one year, inclusive.

Example

user1 is assigned to the ad_hoc_queries resource pool:

```sql
=> CREATE USER user1 RESOURCE POOL ad_hoc_queries;
```

RUNTIMECAP for user1 is set to 1 hour:

```sql
=> ALTER USER user1 RUNTIMECAP '60 minutes';
```

RUNTIMECAP for the ad_hoc_queries resource pool is set to 30 minutes:

```sql
=> ALTER RESOURCE POOL ad_hoc_queries RUNTIMECAP '30 minutes';
```

In this example, Vertica terminates user1's queries if they exceed 30 minutes. Although the user1's runtime limit is set to one hour, the pool on which the query runs, which has a 30-minute runtime limit, has precedence.

Note: If a secondary pool for the ad_hoc_queries pool is specified using the CASCADE TO function, the query executes on that pool when the RUNTIMECAP on the ad_hoc_queries pool is surpassed.
Handling Session Socket Blocking

A session socket can be blocked while awaiting client input or output for a given query. Session sockets are typically blocked for numerous reasons—for example, when the Vertica execution engine transmits data to the client, or a COPY LOCAL operation awaits load data from the client.

In rare cases, a session socket can remain indefinitely blocked. For example, a query times out on the client, which tries to forcibly cancel the query, or relies on the session RUNTIMECAP setting to terminate it. In either case, if the query ends while awaiting messages or data, the socket can remain blocked and the session hang until it is forcibly closed.

Configuring a Grace Period

You can configure the system with a grace period, during which a lagging client or server can catch up and deliver a pending response. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated. If no grace period is set, the query can maintain its block on the socket indefinitely.

You should set the session grace period high enough to cover an acceptable range of latency and avoid closing sessions prematurely—for example, normal client-side delays in responding to the server. Very large load operations might require you to adjust the session grace period as needed.

You can set the grace period at four levels, listed in descending order of precedence:

1. Session (highest)
2. User
3. Node
4. Database
Setting Grace Periods for the Database and Nodes

At the database and node levels, you set the grace period to any `interval` up to 20 days, through configuration parameter `BlockedSocketGracePeriod`:

- `ALTER DATABASE db-name SET BlockedSocketGracePeriod = 'interval'`;
- `ALTER NODE node-name SET BlockedSocketGracePeriod = 'interval'`;

By default, the grace period for both levels is set to an empty string, which allows unlimited blocking.

Setting Grace Periods for Users and Sessions

You can set the grace period for individual users and for a given session, as follows:

- `{ CREATE | ALTER USER } user-name GRACEPERIOD { 'interval' | NONE };`
- `SET SESSION GRACEPERIOD { 'interval' | = DEFAULT | NONE };`

A user can set a session to any interval equal to or less than the grace period set for that user. Superusers can set the grace period for other users, and for their own sessions, to any value up to 20 days, inclusive.

Examples

Superuser `dbadmin` sets the database grace period to 6 hours. This limit only applies to non-superusers. `dbadmin` can set the session grace period for herself to any value up to 20 days—in this case, 10 hours:

```sql
=> ALTER DATABASE VMart SET BlockedSocketGracePeriod = '6 hours';
ALTER DATABASE
=> SHOW CURRENT BlockedSocketGracePeriod;
level | name | setting
-----------------------------
DATABASE | BlockedSocketGracePeriod | 6 hours
(1 row)

=> SET SESSION GRACEPERIOD '10 hours';
SET
=> SHOW GRACEPERIOD;
name | setting
-----------------------------
graceperiod | 10:00
(1 row)
```
dbadmin creates user user777 created with no grace period setting. Thus, the effective grace period for user777 is derived from the database setting of BlockedSocketGracePeriod, which is 6 hours. Any attempt by user777 to set the session grace period to a value greater than 6 hours returns with an error:

```
=> CREATE USER user777;
=> \c - user777
You are now connected as user "user777".
=> SHOW GRACEPERIOD;
   name | setting
----------+--------
 graceperiod | 06:00
(1 row)
=> SET SESSION GRACEPERIOD '7 hours';
ERROR 8175: The new period 07:00 would exceed the database limit of 06:00
```

dbadmin sets a grace period of 5 minutes for user777. Now, user777 can set the session grace period to any value equal to or less than the user-level setting:

```
=> \c
You are now connected as user "dbadmin".
=> ALTER USER user777 GRACEPERIOD '5 minutes';
ALTER USER
=> \c - user777
You are now connected as user "user777".
=> SET SESSION GRACEPERIOD '6 minutes';
ERROR 8175: The new period 00:06 would exceed the user limit of 00:05
=> SET SESSION GRACEPERIOD '4 minutes';
SET
```
Using User-Defined Pools and User-Profiles for Workload Management

The scenarios in this section describe common workload-management issues, and provide solutions with examples.

Periodic Batch Loads
Scenario

You do batch loads every night, or occasionally (infrequently) during the day. When loads are running, it is acceptable to reduce resource usage by queries, but at all other times you want all resources to be available to queries.
Solution

Create a separate resource pool for loads with a higher priority than the preconfigured setting on the build-in GENERAL pool.

In this scenario, nightly loads get preference when borrowing memory from the GENERAL pool. When loads are not running, all memory is automatically available for queries.

**Note:** If you are using the WOS, tune the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads. This ensures that AUTO spill to ROS is configured in an optimal fashion.
Example

Create a resource pool with the PRIORITY of the pool set higher than the GENERAL pool.

For example, to create a pool designated for loads that has a higher priority then the GENERAL pool, set load_pool with a priority of 10:

```sql
=> CREATE RESOURCE POOL load_pool PRIORITY 10;
```

Edit the WOSDATA pool PLANNEDCONCURRENCY:

```sql
=> ALTER RESOURCE POOL WOSDATA PLANNEDCONCURRENCY 6;
```

Modify the user's resource pool:

```sql
=> ALTER USER load_user RESOURCE POOL load_pool;
```

CEO Query
Scenario

The CEO runs a report every Monday at 9AM, and you want to be sure that the report always runs.
Solution

To ensure that a certain query or class of queries always gets resources, you could create a dedicated pool for it as follows:

1. Using the `PROFILE` command, run the query that the CEO runs every week to determine how much memory should be allocated:

   ```
   => PROFILE SELECT DISTINCT s.product_key, p.product_description
     FROM store.store_sales_fact s, public.product_dimension p
     WHERE s.product_key = p.product_key AND s.product_version = p.product_version
     AND s.store_key IN (
       SELECT store_key FROM store.store_dimension
       WHERE store_state = 'MA')
     ORDER BY s.product_key;
   ```

2. At the end of the query, the system returns a notice with resource usage:

   ```
   NOTICE: Statement is being profiled. HINT: select * from v_monitor.execution_engine_profiles
   where
   transaction_id=4503596273751349 and statement_id=6;
   NOTICE: Initiator memory estimate for query: [on pool general: 1723648 KB, minimum: 355920 KB]
   ```

3. Create a resource pool with `MEMORYSIZE` reported by the above hint to ensure that the CEO query has at least this memory reserved for it:

   ```
   => CREATE RESOURCE POOL ceo_pool MEMORYSIZE '1800M' PRIORITY 10;
   ```

   Expanded display is on.

   ```
   => SELECT * FROM resource_pools WHERE name = 'ceo_pool';
   ```

<table>
<thead>
<tr>
<th>name</th>
<th>ceo_pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_internal</td>
<td>f</td>
</tr>
<tr>
<td>memorysize</td>
<td>1800M</td>
</tr>
<tr>
<td>maxmemorysize</td>
<td>10</td>
</tr>
<tr>
<td>priority</td>
<td>300</td>
</tr>
<tr>
<td>queuetimeout</td>
<td>4</td>
</tr>
<tr>
<td>plannedconcurrency</td>
<td>4</td>
</tr>
<tr>
<td>maxconcurrency</td>
<td>f</td>
</tr>
<tr>
<td>singleinitiator</td>
<td>f</td>
</tr>
</tbody>
</table>
   ```

4. Assuming the CEO report user already exists, associate this user with the above resource pool using `ALTER USER` statement.

   ```
   => ALTER USER ceo_user RESOURCE POOL ceo_pool;
   ```
5. Issue the following command to confirm that the ceo_user is associated with the ceo_pool:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';
-[
   RECORD 1 ]-+-----------------------------+
   user_id | 45035996273713548
   user_name | ceo_user
   is_super_user | f
   resource_pool | ceo_pool
   memory_cap_kb | unlimited
```

If the CEO query memory usage is too large, you can ask the Resource Manager to reduce it to fit within a certain budget. See Target Memory Determination for Queries in Concurrent Environments.

Preventing Runaway Queries
Scenario

Joe, a business analyst often runs big reports in the middle of the day that take up the whole machine's resources. You want to prevent Joe from using more than 100MB of memory, and you want to also limit Joe's queries to run for less than 2 hours.
Solution

User Profiles provides a solution to this scenario. To restrict the amount of memory Joe can use at one time, set a MEMORYCAP for Joe to 100MB using the ALTER USER command. To limit the amount of time that Joe's query can run, set a RUNTIMECAP to 2 hours using the same command. If any query run by Joe takes up more than its cap, Vertica rejects the query.

If you have a whole class of users whose queries you need to limit, you can also create a resource pool for them and set RUNTIMECAP for the resource pool. When you move these users to the resource pool, Vertica limits all queries for these users to the RUNTIMECAP you specified for the resource pool.
Example

=> ALTER USER analyst_user MEMORYCAP '100M' RUNTIMECAP '2 hours';

If Joe attempts to run a query that exceeds 100MB, the system returns an error that the request exceeds the memory session limit, such as the following example:

```
\i vmart_query_04.sqlvsql:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan on pool general [Request exceeds memory session limit: 137669KB > 102400KB]
```

Only the system database administrator (dbadmin) can increase only the MEMORYCAP setting. Users cannot increase their own MEMORYCAP settings and will see an error like the following if they attempt to edit their MEMORYCAP or RUNTIMECAP settings:

```
ALTER USER analyst_user MEMORYCAP '135M';
ROLLBACK: permission denied
```

Restricting Resource Usage of Ad Hoc Query Application
Scenario

You recently made your data warehouse available to a large group of users who are not experienced SQL users. Some of the users run reports that operate on a large number of rows and overwhelm the system. You want to throttle usage of the system by these users.
Solution

The simplest solution is to create a standalone resource pool for the ad hoc applications so that the total MEMORYSIZE is fixed. Recall that in a standalone pool, MAXMEMORYSIZE is set equal to MEMORYSIZE so no memory can be borrowed from the GENERAL pool. Associate this user pool with database users from which the application uses to connect to the database. Also set RUNTIMECAP to limit the maximum duration of an ad hoc query.

Other solutions include limiting the memory usage of individual users such as in the Preventing Runaway Queries.
Example

To create a standalone resource pool for the ad hoc users, set the MEMORYSIZE equal to the MAXMEMORYSIZE:

```sql
-> CREATE RESOURCE POOL adhoc_pool MEMORYSIZE '200M' MAXMEMORYSIZE '200M' PRIORITY 0 QUEUETIMEOUT 300 PLANNEDCONCURRENCY 4;
-> SELECT pool_name, memory_size_kb, queueing_threshold_kb
    FROM V_MONITOR.RESOURCE_POOL_STATUS w
    WHERE is_standalone = 'true' AND is_internal = 'false';
```

<table>
<thead>
<tr>
<th>pool_name</th>
<th>memory_size_kb</th>
<th>queueing_threshold_kb</th>
</tr>
</thead>
<tbody>
<tr>
<td>adhoc_pool</td>
<td>204800</td>
<td>153600</td>
</tr>
</tbody>
</table>

(1 row)

After the pool has been created, associate the ad hoc users with the adhoc_pool:

```sql
-> ALTER USER app1_user RESOURCE POOL adhoc_pool;
-> ALTER RESOURCE POOL adhoc_pool MEMORYSIZE '10M' MAXMEMORYSIZE '10M';
```

The query will not borrow memory from the GENERAL pool and gets rejected with a 'Request Too Large' message.

Setting a Hard Limit on Concurrency for an Application
Scenario

For billing purposes, analyst Jane would like to impose a hard limit on concurrency for this application. How can she achieve this?
Solution

The simplest solution is to create a separate resource pool for the users of that application and set its MAXCONCURRENCY to the desired concurrency level. Any queries beyond MAXCONCURRENCY are queued.

Tip: Vertica recommends leaving PLANNEDCONCURRENCY to the default level so the queries get their maximum amount of resources. The system as a whole thus runs with the highest efficiency.
Example

In this example, there are four billing users associated with the billing pool. The objective is to set a hard limit on the resource pool so a maximum of three concurrent queries can be executed at one time. All other queries will queue and complete as resources are freed.

=> CREATE RESOURCE POOL billing_pool MAXCONCURRENCY 3 QUEUETIMEOUT 2;
=> CREATE USER bill1_user RESOURCE POOL billing_pool;
=> CREATE USER bill2_user RESOURCE POOL billing_pool;
=> CREATE USER bill3_user RESOURCE POOL billing_pool;
=> CREATE USER bill4_user RESOURCE POOL billing_pool;
=> \x

Expanded display is on.

=> select maxconcurrency,queuetimeout from resource_pools where name = 'billing_pool';
maxconcurrency | queuetimeout
----------------- | ------------
            3 |         2

(1 row)

> SELECT reason, resource_type, rejection_count FROM RESOURCE_REJECTIONS
WHERE pool_name = 'billing_pool' AND node_name ilike '%node0001';

reason | resource_type | rejection_count
----------------------- | ------------ | ------------
Timedout waiting for resource request | Queries | 16

(1 row)

If queries are running and do not complete in the allotted time (default timeout setting is 5 minutes), the next query requested gets an error similar to the following:

ERROR: Insufficient resources to initiate plan on pool billing_pool [Timedout waiting for resource request: Request exceeds limits: Queries Exceeded: Requested = 1, Free = 0 (Limit = 3, Used = 3)]

The table below shows that there are three active queries on the billing pool.

=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb FROM RESOURCE_ACQUISITIONS
WHERE pool_name = 'billing_pool';

pool_name | thread_count | open_file_handle_count | memory_inuse_kb
---------------------- | ------------ | ---------------------- | ---------------
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870

(3 rows)

Handling Mixed Workloads: Batch versus Interactive
Scenario

You have a web application with an interactive portal. Sometimes when IT is running batch reports, the web page takes a long time to refresh and users complain, so you want to provide a better experience to your web site users.
Solution

The principles learned from the previous scenarios can be applied to solve this problem. The basic idea is to segregate the queries into two groups associated with different resource pools. The prerequisite is that there are two distinct database users issuing the different types of queries. If this is not the case, do consider this a best practice for application design.

Method 1
Create a dedicated pool for the web page refresh queries where you:

- Size the pool based on the average resource needs of the queries and expected number of concurrent queries issued from the portal.

- Associate this pool with the database user that runs the web site queries. See CEO Query for information about creating a dedicated pool.

This ensures that the web site queries always run and never queue behind the large batch jobs. Leave the batch jobs to run off the GENERAL pool.

For example, the following pool is based on the average resources needed for the queries running from the web and the expected number of concurrent queries. It also has a higher PRIORITY to the web queries over any running batch jobs and assumes the queries are being tuned to take 250M each:

```sql
=> CREATE RESOURCE POOL web_pool
    MEMORYSIZE '250M'
    MAXMEMORYSIZE NONE
    PRIORITY 10
    MAXCONCURRENCY 5
    PLANNEDCONCURRENCY 1;
```

Method 2
Create a standalone pool to limit the batch reports down to a fixed memory size so memory is always left available for other purposes. For details, see Restricting Resource Usage of Ad Hoc Query Application.

For example:

```sql
=> CREATE RESOURCE POOL batch_pool
    MEMORYSIZE '4G'
    MAXMEMORYSIZE '4G'
    MAXCONCURRENCY 10;
```

The same principle can be applied if you have three or more distinct classes of workloads.

Setting Priorities on Queries Issued By Different Users
Scenario

You want user queries from one department to have a higher priority than queries from another department.
Solution

The solution is similar to the mixed workload case. In this scenario, you do not limit resource usage; you set different priorities. To do so, create two different pools, each with MEMORYSIZE=0% and a different PRIORITY parameter. Both pools borrow from the GENERAL pool, however when competing for resources, the priority determine the order in which each pool's request is granted. For example:

```sql
=> CREATE RESOURCE POOL dept1_pool PRIORITY 5;
=> CREATE RESOURCE POOL dept2_pool PRIORITY 8;
```

If you find this solution to be insufficient, or if one department's queries continuously starves another department’s users, you can add a reservation for each pool by setting MEMORYSIZE so some memory is guaranteed to be available for each department.

For example, both resources use the GENERAL pool for memory, so you can allocate some memory to each resource pool by using ALTER RESOURCE POOL to change MEMORYSIZE for each pool:

```sql
=> ALTER RESOURCE POOL dept1_pool MEMORYSIZE '100M';
=> ALTER RESOURCE POOL dept2_pool MEMORYSIZE '150M';
```

Continuous Load and Query
Scenario

You want your application to run continuous load streams, but many have up concurrent query streams. You want to ensure that performance is predictable.
Solution

The solution to this scenario depends on your query mix. In all cases, the following approach applies:

1. Determine the number of continuous load streams required. This may be related to the desired load rate if a single stream does not provide adequate throughput, or may be more directly related to the number of sources of data to load. Also determine if automatic storage is best, or if DIRECT is required. Create a dedicated resource pool for the loads, and associate it with the database user that will perform them. See CREATE RESOURCE POOL for details.

   In general, concurrency settings for the load pool should be less than the number of cores per node. Unless the source processes are slow, it is more efficient to dedicate more memory per load, and have additional loads queue. Adjust the load pool's QUEUETIMEOUT setting if queuing is expected.

2. If using automatic targeting of COPY and INSERT, set the PLANNEDCONCURRENCY parameter of the WOSDATA pool to the number of concurrent loads expected. Also, set MEMORYSIZE of the WOS to the expected size of the loaded data to ensure that small loads don't spill to ROS immediately. See Built-In Pools for details.

3. Run the load workload for a while and observe whether the load performance is as expected. If the Tuple Mover is not tuned adequately to cover the load behavior, see Managing the Tuple Mover in Administrator's Guide.

4. If there is more than one kind of query in the system—for example, some queries must be answered quickly for interactive users, while others are part of a batch reporting process—follow the guidelines in Handling Mixed Workloads: Batch versus Interactive.

5. Let the queries run and observe performance. If some classes of queries do not perform as desired, then you might need to tune the GENERAL pool as outlined in Restricting Resource Usage of Ad Hoc Query Application, or create more dedicated resource pools for those queries. See more information, see CEO Query and Handling Mixed Workloads: Batch versus Interactive.

See the sections on Managing Workloads and CREATE RESOURCE POOL for information on obtaining predictable results in mixed workload environments.

Prioritizing Short Queries at Run Time
Scenario

You recently created a resource pool for users who are not experienced with SQL and who frequently run ad hoc reports. You have managed resource allocation by creating a standalone resource pool that will prevent these queries for borrowing resources from the GENERAL pool, but now you want to manage resources at run time and ensure that short queries always run with a high priority and are never queued as a result of limited run-time resources.
Solution

Set the RUNTIMEPRIORITY for the resource pool to MEDIUM or LOW. Set the RUNTIMEPRIORITYTHRESHOLD for the resource pool to the duration of queries you want to ensure always run at a high priority. For instance, if you set this value to 5, all queries that complete within 5 seconds will run at high priority. Any other query that exceeds 5 seconds will drop down to the RUNTIMEPRIORITY assigned to the resource pool (MEDIUM or LOW).
Example

To ensure that all queries with a duration of less than 5 seconds always run at a high priority, modify adhoc_pool as follows:

- Set the RUNTIMEPRIORITY to MEDIUM
- Set the RUNTIMETHRESHOLD to 5

```sql
=> ALTER RESOURCE POOL adhoc_pool RUNTIMEPRIORITY medium RUNTIMEPRIORITYTHRESHOLD 5;
```

Dropping the Runtime Priority of Long Queries
Scenario

You want most queries in a resource pool to run at a HIGH runtime priority; however, you'd like to be able to drop jobs longer than 1 hour to a lower priority.
Solution

Set the RUNTIMEPRIORITY for the resource pool to LOW and set the RUNTIMEPRIORITYTHRESHOLD to a number that cuts off only the longest jobs.
Example

To ensure that all queries with a duration of more than 3600 seconds (1 hour) are assigned a low runtime priority, modify the resource pool as follows:

- Set the RUNTIMEPRIORITY to LOW.
- Set the RUNTIMETHRESHOLD to 3600

```sql
=> ALTER RESOURCE POOL ad_hoc_pool RUNTIMEPRIORITY low RUNTIMEPRIORITYTHRESHOLD 3600;
```
Tuning Built-In Pools

The scenarios in this section describe how to tune built-in pools.

- Restricting Vertica to Take Only 60% of Memory
- Tuning for Recovery
- Tuning for Refresh
- Tuning Tuple Mover Pool Settings
- Tuning for Machine Learning

Restricting Vertica to Take Only 60% of Memory
Scenario

You have a single node application that embeds Vertica, and some portion of the RAM needs to be devoted to the application process. In this scenario, you want to limit Vertica to use only 60% of the available RAM.
Solution

Set the MAXMEMORYSIZE parameter of the GENERAL pool to the desired memory size. See Resource Pool Architecture for a discussion on resource limits.

Tuning for Recovery
Scenario

You have a large database that contains a single large table with two projections, and with default settings, recovery is taking too long. You want to give recovery more memory to improve speed.
Solution

Set the PLANNEDCONCURRENCY and MAXCONCURRENCY setting of the recovery pool to 1 so that recovery can take as much memory as possible from the GENERAL pool and run only one thread at once.

Note: This setting could slow down other queries in your system.

Tuning for Refresh
Scenario

When a refresh operation is running, system performance is affected and user queries get rejected. You want to reduce the memory usage of the refresh job.
**Solution**

Set the MEMORYSIZE parameter of the refresh pool to a fixed value. The Resource Manager then tunes the refresh query to only use this amount of memory.

Tip: Remember to reset the refresh pool MEMORYSIZE back to 0% after the refresh operation completes so memory can be used for other operations.

**Tuning Tuple Mover Pool Settings**
Scenario

During loads, you occasionally notice spikes in the number of ROS containers, and you would like to make the Tuple Mover more aggressive.
Solution

Increase the MAXCONCURRENCY parameter of the TM pool to 3 or higher. This setting ensures that the Tuple Mover can run more than one mergeout thread, so if a large mergeout is in progress, smaller ROS containers can also be merged, thus preventing a buildup of ROS containers.

Tuning for Machine Learning
Scenario

You have a large number of machine learning algorithms running, and you want to improve the performance of the queries. In this scenario, you want to give machine learning functions more memory to improve their performance.
Solution

Vertica executes the machine learning functions in the BLOBDATA resource pool. To improve the performance of the machine learning functions increase the MAXMEMORYSIZE parameter of the BLOBDATA pool. If a query using the BLOBDATA pool exceeds its query planning budget, then it spills to disk.

For more information about tuning your query budget, see Target Memory Determination for Queries in Concurrent Environments.
See Also

- ALTER RESOURCE POOL
- Managing Resources At Query Run Time
- Built-In Pool Configuration
- Built-In Pools

Reducing Query Run Time

The run time of queries depends on the complexity of the query, the number of operators in the plan, data volumes, and projection design. If the system is bottlenecked on either I/O or CPU, queries could run more slowly than expected. In most cases, high CPU usage can be alleviated by better projection design, and high I/O is usually due to contention because of operations like joins and sorts that spill to disk. However, there is no single solution to fix high CPU or high I/O usage, so queries must be examined and tuned individually.

You can evaluate a slow-running query in two ways:

- Prefix the query with EXPLAIN to view the optimizer's query plan.
- Examine the execution profile by querying the system table EXECUTION_ENGINPROFILES

Examining the query plan often reveals one or more of the following:

- Suboptimal projection sort order
- Predicate evaluation on an unsorted or unencoded column
- Use of GROUPBY HASH instead of GROUPBY PIPE

See Creating Custom Designs to understand projection design techniques. The Database Designer automatically applies these techniques to suggest optimal designs for queries.

Profiling

Vertica provides profiling mechanisms that let you determine how well the database is performing. For example, Vertica can collect profiling data for a single statement, a single session, or for all sessions on all nodes. For details, see Profiling Database Performance.
Managing System Resource Usage

You can use the Using System Tables to track overall resource usage on your cluster. These and the other system tables are described in the Vertica System Tables.

If your queries are experiencing errors due to resource unavailability, you can use the following system tables to obtain more details:

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCE_REJECTIONS</td>
<td>Monitors requests for resources that are rejected by the Resource Manager.</td>
</tr>
<tr>
<td>DISK_RESOURCE_REJECTIONS</td>
<td>Monitors requests for resources that are rejected due to disk space shortages. See Managing Disk Space for more information.</td>
</tr>
</tbody>
</table>

When requests for resources of a certain type are being rejected, do one of the following:

- Increase the resources available on the node by adding more memory, more disk space, and so on. See Managing Disk Space.
- Reduce the demand for the resource by reducing the number of users on the system (see Managing Sessions), rescheduling operations, and so on.

The LAST_REJECTED_VALUE field in RESOURCE_REJECTIONS indicates the cause of the problem. For example:

- The message Usage of a single requests exceeds high limit means that the system does not have enough of the resource available for the single request. A common example occurs when the file handle limit is set too low and you are loading a table with a large number of columns.
- The message Timed out or Canceled waiting for resource reservation usually means that there is too much contention for the resource because the hardware platform cannot support the number of concurrent users using it.

Managing Sessions

Vertica provides several methods for database administrators to view and control sessions. The methods vary according to the type of session:
External (user) sessions are initiated by vsq1 or programmatic (ODBC or JDBC) connections and have associated client state.

Internal (system) sessions are initiated by Vertica and have no client state.

Configuring Maximum Sessions

The maximum number of per-node user sessions is set by the configuration parameter MaxClientSessions parameter, by default 50. You can set MaxClientSessions parameter to any value between 0 and 1000. In addition to this maximum, Vertica also allows up to five administrative sessions per node.

For example:

```sql
=> ALTER DATABASE mydb SET MaxClientSessions = 100;
```

Note: If you use the Administration Tools "Connect to Database" option, Vertica will attempt connections to other nodes if a local connection does not succeed. These cases can result in more successful "Connect to Database" commands than you would expect given the MaxClientSessions value.

Viewing Sessions

The system table SESSIONS contains detailed information about user sessions and returns one row per session. Superusers have unrestricted access to all database metadata. Access for other users varies according to their privileges.

Interrupting and Closing Sessions

You can interrupt a running statement with the Vertica function INTERRUPT_STATEMENT. Interrupting a running statement returns a session to an idle state:

- No statements or transactions are running.
- No locks are held.
- The database is doing no work on behalf of the session.

Closing a user session interrupts the session and disposes of all state related to the session, including client socket connections for the target sessions. The following Vertica functions close one or more user sessions:
CLOSE_SESSION

CLOSE_ALL_SESSIONS

CLOSE_USER_SESSIONS

SHUTDOWN

SELECT statements that call these functions return after the interrupt or close message is delivered to all nodes. The function might return before Vertica completes execution of the interrupt or close operation. Thus, there might be a delay after the statement returns and the interrupt or close takes effect throughout the cluster. To determine if the session or transaction ended, query the SESSIONS system table.

In order to shut down a database, you must first close all user sessions. For more about database shutdown, see Stopping the Database.

Managing Load Streams

You can use the Using System Tables to keep track of data being loaded on your cluster.

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD_STREAMS</td>
<td>Monitors load metrics for each load stream on each node.</td>
</tr>
</tbody>
</table>

For details about Vertica system tables, see Vertica System Tables.

When a COPY statement using the DIRECT option is in progress, the ACCEPTED_ROW_COUNT value can increase during parsing. This value can reach the maximum number of rows in the input file.

If COPY reads input data from multiple named pipes, the PARSE_COMPLETE_PERCENT value remains at zero (0) until all named pipes return an EOF. While COPY awaits an EOF from multiple pipes, it can appear to be hung. However, before canceling the COPY statement, check your system CPU and disk accesses to determine if any activity is in progress.

In a typical load, the PARSE_COMPLETE_PERCENT value can either increase slowly or jump quickly to 100%, if you are loading from named pipes or STDIN. However, SORT_COMPLETE_PERCENT remains at 0 when loading from named pipes or STDIN. After PARSE_COMPLETE_PERCENT reaches 100%, SORT_COMPLETE_PERCENT increases to 100%. Depending on the data sizes, a significant lag can occur between the time PARSE_COMPLETE_PERCENT reaches 100% sand the time SORT_COMPLETE_PERCENT begins to increase.
Monitoring Vertica

This section describes some of the ways in which you can monitor the health of your database. You can also monitor the Vertica database using MC, as described in Monitoring Using MC.

Monitoring Log Files

When a Database Is Running

When a Vertica database is running, each node in the cluster writes messages into a file named vertica.log. For example, the Tuple Mover and the transaction manager write INFO messages into vertica.log at specific intervals even when there is no WOS activity.

You configure the location of the vertica.log file. By default, the log file is in catalog-path/database-name/node-name_catalog/vertica.log where

- `catalog-path` is the path shown in the NODES system table minus the Catalog directory at the end.
- `database-name` is the name of your database.
- `node-name_catalog` is the name of the node shown in the NODES system table.

Note: Vertica often changes the format or content of log files in subsequent releases to benefit both customers and customer support.

To monitor one node in a running database in real time:

1. Log in to the database administrator account on any node in the cluster.
2. In a terminal window enter:

   $ tail -f catalog-path/database-name/node-name_catalog/vertica.log

Note: To monitor your overall database (rather than an individual node/host), use the Data Collector, which records system activities and performance. See Retaining Monitoring Information for more on Data Collector.
### catalog-path
The catalog pathname specified when you created the database. See Creating a Database in the Administrator's Guide.

### database-name
The database name (case sensitive)

### node-name
The node name, as specified in the database definition. See Viewing a Database in the Administrator's Guide.

## When the Database / Node Is Starting up

During system startup, before the Vertica log has been initialized to write messages, each node in the cluster writes messages into a file named dbLog. This log is useful to diagnose situations where the database fails to start before it can write messages into vertica.log. The dblog is located at the following path, using catalog-path and database-name as described above:

```
catalog-path/database-name/dbLog
```

## See Also

- Rotating Log Files

## Rotating Log Files

Most Linux distributions include the logrotate utility. Using this utility simplifies log file administration. By setting up a logrotate configuration file, you can use the utility to complete one or more of these tasks automatically:

- Compress and rotate log files
- Remove log files automatically
- Email log files to named recipients

You can configure logrotate to complete these tasks at specific intervals, or when log files reach a particular size.
If `logrotate` is present when Vertica is installed, then Vertica automatically sets this utility to look for configuration files. Thus, `logrotate` searches for configuration files in the `/opt/vertica/config/logrotate` directory on each node.

When you create a database, Vertica creates database-specific `logrotate` configurations on each node in your cluster, which are used by the `logrotate` utility. It then creates a file with the path `/opt/vertica/config/logrotate/<dbname>` for each individual database.

For information about additional settings, use the `man logrotate` command.

### Executing the Python script Through the dbadmin logrotate cron Job

During the installation of Vertica, the installer configures a cron job for the `dbadmin` user. This cron job is configured to execute a Python script that runs the `logrotate` utility. You can view the details of this cron job by viewing the `dbadmin.cron` file, which is located in the `/opt/vertica/config` directory.

If you want to customize a cron job to configure `logrotate` for your Vertica database, you must create the cron job under the `dbadmin` user.

### Using the Administration Tools Logrotate Utility

You can use the `admintools logrotate` option to help configure `logrotate` scripts for a database and distribute the scripts across the cluster. The `logrotate` option allows you to specify:

- How often to rotate logs
- How large logs can become before being rotated
- How long to keep the logs

**Example:**

The following example shows you how to set up log rotation on a weekly schedule and keeps for three months (12 logs).

```bash
$ admintools -t logrotate -d <dbname> -r weekly -k 12
```

See [Writing Administration Tools Scripts](#) for more usage information.
Rotating Logs Manually

To implement a custom log rotation process, follow these steps:

1. Rename or archive the existing `vertica.log` file. For example:

   ```bash
   $ mv vertica.log vertica.log.1
   ```

2. Send the Vertica process the USR1 signal, using either of the following approaches:

   ```bash
   $ killall -USR1 vertica
   or
   $ ps -ef | grep -i vertica
   $ kill -USR1 process-id
   ```

See Also

- Monitoring Log Files
Monitoring Process Status (ps)

You can use `ps` to monitor the database and Spread processes running on each node in the cluster. For example:

```
$ ps aux | grep /opt/vertica/bin/vertica
$ ps aux | grep /opt/vertica/sbin/spread
```

You should see one Vertica process and one Spread process on each node for common configurations. To monitor Administration Tools and connector processes:

```
$ ps aux | grep vertica
```

There can be many connection processes but only one Administration Tools process.
Monitoring Linux Resource Usage

You should monitor system resource usage on any or all nodes in the cluster. You can use System Activity Reporting (SAR) to monitor resource usage.

[[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you install pstack and sysstat to help monitor Linux resources. The SYSSTAT package contains utilities for monitoring system performance and usage activity, such as sar, as well as tools you can schedule via cron to collect performance and activity data. See the SYSSTAT Web page for details.

The pstack utility lets you print a stack trace of a running process. See the PSTACK man page for details.

1. Log in to the database administrator account on any node.

2. Run the top utility

   $ top

   A high CPU percentage in top indicates that Vertica is CPU-bound. For example:

   ```
   top - 11:44:28 up 53 days, 23:47, 9 users, load average: 0.91, 0.97, 0.81
   Tasks: 123 total, 1 running, 122 sleeping, 0 stopped, 0 zombie
   Cpu(s): 26.9%us, 1.3%sy, 0.0%ni, 71.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
   Mem: 4053136 total, 3882020k used, 171116 free, 407688 buffers
   Swap: 4192956 total, 176k used, 4192780 free, 1526436 cached
   PID USER      PR NI VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
   13703 dbadmin  20  0  33280  12m  217m S   0.1 0.0   0:00.00  sqlserver
   8501 dbadmin  20  0  38080  20m  266m S   0.0 0.0   0:00.00  sqlserver
   1 root        20  0  49640  24m  455m S   0.0 0.0   0:00.00  ksoftirqd/0
   ...
   ```

   Some possible reasons for high CPU usage are:
   
   - The Tuple Mover runs automatically and thus consumes CPU time even if there are no connections to the database.

   - Log in as root and change the Linux parameter swappiness to 1.
# echo 1 > /proc/sys/vm/swappiness

Setting swappiness in this manner does not save the value after a reboot. To permanently set the swappiness value, add or update the following in /etc/sysctl.conf:

```
vm.swappiness = 1
```

You can also check the swappiness value as follows:

```
cat /proc/sys/vm/swappiness
```

- Some information sources:
  - Red Hat
  - Indiana University Unix Systems Support Group

3. Run the `iostat` utility. A high idle time in `top` at the same time as a high rate of blocks read in `iostat` indicates that Vertica is disk-bound. For example:

```
$ /usr/bin/iostat
Linux 2.6.18-164.el5 (qa01) 02/05/2011
avg-cpu: %user %nice %system %iowait %steal %idle
                 2.32  0.76  0.68  0.00  95.47
Device: tps Blk_read/s Blk_wrtn/s Blk_read Blk_wrtn
hda    0.37   3.40    19.37  2117723  6464640
sda    0.46   1.94    18.96  1208130  11816472
sdb    0.26   1.79    15.69  1114792  9781840
sdc    0.24   1.80    16.06  1119304 10010328
sdd    0.22   1.79    15.52  1117472  9676200
md0    8.37   7.31    66.23 4554834 41284840
```

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

<table>
<thead>
<tr>
<th>System table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK_STORAGE</td>
<td>Monitors the amount of disk storage used by the database on each node.</td>
</tr>
<tr>
<td>COLUMN_STORAGE</td>
<td>Monitors the amount of disk storage used by each column of each projection on each node.</td>
</tr>
<tr>
<td>System table</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROJECTION_STORAGE</td>
<td>Monitors the amount of disk storage used by each projection on each node.</td>
</tr>
</tbody>
</table>

**Monitoring Database Size for License Compliance**

Your Vertica license can include a data storage allowance. The allowance can consist of data in columnar tables, flex tables, or both types of data. The `AUDIT()` function estimates the columnar table data size and any flex table materialized columns. The `AUDIT_FLEX()` function estimates the amount of `__raw__` column data in flex or columnar tables. In regards to license data limits, data in `__raw__` columns is calculated at 1/10th the size of structured data. Monitoring data sizes for columnar and flex tables lets you plan either to schedule deleting old data to keep your database in compliance with your license, or to consider a license upgrade for additional data storage.

**Note:** An audit of columnar data includes flex table real and materialized columns, but not `__raw__` column data.

**Viewing Your License Compliance Status**

Vertica periodically runs an audit of the columnar data size to verify that your database is compliant with your license terms. You can view the results of the most recent audit by calling the `GET_COMPLIANCE_STATUS` function.

```sql
=> select GET_COMPLIANCE_STATUS();

GET_COMPLIANCE_STATUS

-----------------------------------------------
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization   : 50%
Audit Time    : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Periodically running `GET_COMPLIANCE_STATUS` to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your
Manually Auditing Columnar Data Usage

You can manually check license compliance for all columnar data in your database using the `AUDIT_LICENSE_SIZE` function. This function performs the same audit that Vertica periodically performs automatically. The `AUDIT_LICENSE_SIZE` check runs in the background, so the function returns immediately. You can then query the results using `GET_COMPLIANCE_STATUS`.

Note: When you audit columnar data, the results include any flex table real and materialized columns, but not data in the `__raw__` column. Materialized columns are virtual columns that you have promoted to real columns. Columns that you define when creating a flex table, or which you add with `ALTER TABLE...ADD COLUMN` statements are real columns. All `__raw__` columns are real columns. However, since they consist of unstructured or semi-structured data, they are audited separately.

An alternative to `AUDIT_LICENSE_SIZE` is to use the `AUDIT` function to audit the size of the columnar tables in your entire database by passing an empty string to the function. This function operates synchronously, returning when it has estimated the size of the database.

```sql
=> SELECT AUDIT('');
AUDIT
---------
76376696
(1 row)
```

The size of the database is reported in bytes. The `AUDIT` function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the `AUDIT` function in the SQL Reference Manual for full details. Vertica does not count the `AUDIT` function results as an official audit. It takes no license compliance actions based on the results.

Note: The results of the `AUDIT` function do not include flex table data in `__raw__` columns. Use the `AUDIT_FLEX` function to monitor data usage flex tables.

Manually Auditing `__raw__` Column Data

You can use the `AUDIT_FLEX` function to manually audit data usage for flex or columnar tables with a `__raw__` column. The function calculates the encoded, compressed data stored in ROS
containers for any Raw columns. Materialized columns in flex tables are calculated by the AUDIT function. The AUDIT_FLEX results do not include data in the Raw columns of temporary flex tables.

**Targeted Auditing**

If audits determine that the columnar table estimates are unexpectedly large, consider schemas, tables, or partitions that are using the most storage. You can use the AUDIT function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the online_sales schema in the VMart example database, run the following command:

```sql
=> SELECT AUDIT('online_sales');
AUDIT
---------
35716504
(1 row)
```

You can also change the granularity of an audit to report the size of each object in a larger entity (for example, each table in a schema) by using the granularity argument of the AUDIT function. See the AUDIT function in the SQL Reference Manual.

**Using Management Console to Monitor License Compliance**

You can also get information about data storage of columnar data (for columnar tables and for materialized columns in flex tables) through the Management Console. This information is available in the database Overview page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.
- The Audit button returns the same information as the AUDIT() function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.
Monitoring Elastic Cluster Rebalancing

Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

- The `REBALANCE_TABLE_STATUS` table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

  **Note:** If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

- `REBALANCE_PROJECTION_STATUS` can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, `separated_percent` and `transferred_percent` can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "where is_latest" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the table_id and "<unknown>" is displayed for the table_name. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.

Monitoring Parameters

The following table describes the monitoring parameters for configuring Vertica.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SnmpTrapDestinationsList</td>
<td>Defines where Vertica sends traps for SNMP. See Configuring Reporting for SNMP. Default Value: none Example: ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';</td>
</tr>
<tr>
<td>SnmpTrapsEnabled</td>
<td>Enables event trapping for SNMP. See Configuring Reporting for SNMP. Default Value: 0 Example: ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;</td>
</tr>
<tr>
<td>SyslogEnabled</td>
<td>Enables event trapping for syslog. See Configuring Reporting for Syslog. Default Value: 0 Example: ALTER DATABASE mydb SET SyslogEnabled = 1;</td>
</tr>
<tr>
<td>SyslogEvents</td>
<td>Defines events that generate a syslog entry. See Configuring Reporting for Syslog. Default Value: none Example: ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';</td>
</tr>
<tr>
<td>SyslogFacility</td>
<td>Defines which SyslogFacility Vertica uses. See Configuring Reporting for Syslog.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **Reporting for Syslog.**  
**Default Value:** user  
**Example:**  
ALTER DATABASE mydb SET SyslogFacility = 'ftp'; |
Monitoring Events

To help you monitor your database system, Vertica traps and logs significant events that affect database performance and functionality if you do not address their root causes. This section describes where events are logged, the types of events that Vertica logs, how to respond to these events, the information that Vertica provides for these events, and how to configure event monitoring.

Event Logging Mechanisms

Vertica posts events to the following mechanisms:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertica.log</td>
<td>All events are automatically posted to vertica.log. See Monitoring the Log Files.</td>
</tr>
<tr>
<td>ACTIVE_EVENTS</td>
<td>This SQL system table provides information about all open events. See Using System Tables and ACTIVE_EVENTS.</td>
</tr>
<tr>
<td>SNMP</td>
<td>To post traps to SNMP, enable global reporting in addition to each individual event you want trapped. See Configuring Event Reporting.</td>
</tr>
<tr>
<td>Syslog</td>
<td>To log events to syslog, enable event reporting for each individual event you want logged. See Configuring Event Reporting.</td>
</tr>
</tbody>
</table>

Event Severity Types

Event names are sensitive to case and spaces. Vertica logs the following events:

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Event Type</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Disk Space</td>
<td>0</td>
<td>The database is running out of disk space or a disk is failing or there is a I/O hardware failure.</td>
<td>It is imperative that you add more disk space or replace the failing disk or hardware as soon as possible.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Event Type</td>
<td>Description</td>
<td>Action</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Read Only File System</td>
<td>1</td>
<td>The database does not have write access to the file system for the data or catalog paths. This can sometimes occur if Linux remounts a drive due to a kernel issue.</td>
<td>Modify the privileges on the file system to give the database write access.</td>
</tr>
<tr>
<td>Loss Of K Safety</td>
<td>2</td>
<td>The database is no longer K-Safe because there are insufficient nodes functioning within the cluster. Loss of K-safety causes the database to shut down. In a four-node cluster, for example, K-safety=1. If one node fails, the fault tolerance is at a critical level. If two nodes fail, the system loses K-safety.</td>
<td>If a system shuts down due to loss of K-safety, you need to recover the system. See Failure Recovery in the Administrator's Guide.</td>
</tr>
<tr>
<td>Current Fault Tolerance at Critical Level</td>
<td>3</td>
<td>One or more nodes in the cluster have failed. If the database loses one more node, it is no longer K-Safe and it shuts down. (For</td>
<td>Restore any nodes that have failed or been shut down.</td>
</tr>
</tbody>
</table>

Check `dmesg` to see what caused the problem.
Also, use the `DISK_RESOURCE_REJECTIONS` system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected. See Managing Disk Space for more information about low disk space.
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Event Type</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Many ROS Containers</td>
<td>4</td>
<td>Due to heavy data load conditions, there are too many ROS containers. This occurs when the Tuple Mover falls behind in performing mergeout operations. The resulting excess number of ROS containers can exhaust all available system resources. To prevent this, Vertica automatically rolls back all transactions that would load data until the Tuple Mover has time to catch up.</td>
<td>You might need to adjust the Tuple Mover's configuration parameters to compensate for the load pattern or rate. See Managing the Tuple Mover in the Administrator's Guide for details. You can query the TUPLE_MOVER_OPERATIONS table to monitor mergeout activity. However, the Tuple Mover does not immediately start a mergeout when a projection reaches the limit of ROS containers, so you may not see a mergeout in progress when receiving this error. If waiting for a mergeout does not resolve the error, the problem probably is related to insufficient RAM. A good rule of thumb is that system RAM in MB divided by 6 times the number of columns in the largest table should be greater than 10. For example, for a 100 column table you would want at least 6GB of RAM (6144MB / (6 * 100) = 10.24) to handle continuous loads.</td>
</tr>
<tr>
<td>WOS Over Flow</td>
<td>5</td>
<td>The WOS cannot hold all the data that you are attempting to load. This means that the</td>
<td>Consider loading the data to disk (ROS) instead of memory (WOS) or splitting the fact</td>
</tr>
<tr>
<td>Event Name</td>
<td>Event Type</td>
<td>Description</td>
<td>Action</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>copy fails and the transaction rolls back.</td>
<td></td>
<td>table load file into multiple pieces and then performing multiple loads in sequence. You might also consider making the Tuple Mover's moveout operation more aggressive. See Managing the Tuple Mover in Administrator's Guide.</td>
<td></td>
</tr>
<tr>
<td>Node State Change</td>
<td>6</td>
<td>The node state has changed.</td>
<td>Check the status of the node.</td>
</tr>
<tr>
<td>Recovery Failure</td>
<td>7</td>
<td>The database was not restored to a functional state after a hardware or software related failure.</td>
<td>The reason for recovery failure can vary. See the event description for more information about your specific situation.</td>
</tr>
<tr>
<td>Recovery Error</td>
<td>8</td>
<td>The database encountered an error while attempting to recover. If the number of recovery errors exceeds Max Tries, the Recovery Failure event is triggered. See Recovery Failure within this table.</td>
<td>The reason for a recovery error can vary. See the event description for more information about your specific situation.</td>
</tr>
<tr>
<td>Recovery Lock Error</td>
<td>9</td>
<td>A recovering node could not obtain an S lock on the table. If you have a continuous stream of COPY commands in progress, recovery might not be able to obtain this lock even after multiple re-tries.</td>
<td>Either momentarily stop the loads or pick a time when the cluster is not busy to restart the node and let recovery proceed.</td>
</tr>
<tr>
<td>Recovery Projection Retrieval Error</td>
<td>10</td>
<td>Vertica was unable to retrieve information about a projection.</td>
<td>The reason for a recovery projection retrieval error can vary. See the event description</td>
</tr>
<tr>
<td>Event Name</td>
<td>Event Type</td>
<td>Description</td>
<td>Action</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Refresh Error</td>
<td>11</td>
<td>The database encountered an error while attempting to refresh.</td>
<td>The reason for a refresh error can vary. See the event description for more information about your specific situation.</td>
</tr>
<tr>
<td>Refresh Lock Error</td>
<td>12</td>
<td>The database encountered a locking error during refresh.</td>
<td>The reason for a refresh error can vary. See the event description for more information about your specific situation.</td>
</tr>
<tr>
<td>Tuple Mover Error</td>
<td>13</td>
<td>The database encountered an error while attempting to move the contents of the Write Optimized Store (WOS) into the Read Optimized Store (ROS).</td>
<td>The reason for a Tuple Mover error can vary. See the event description for more information about your specific situation.</td>
</tr>
<tr>
<td>Timer Service Task Error</td>
<td>14</td>
<td>An error occurred in an internal scheduled task.</td>
<td>Internal use only</td>
</tr>
<tr>
<td>Stale Checkpoint</td>
<td>15</td>
<td>Data in the WOS has not been completely moved out in a timely manner. An UNSAFE shutdown could require reloading a significant amount of data.</td>
<td>Be sure that Moveout operations are executing successfully. Check the vertica.log files for errors related to Moveout.</td>
</tr>
<tr>
<td>CRC Mismatch</td>
<td>16</td>
<td>The Cyclic Redundancy Check returned an error or errors while fetching data.</td>
<td>Review the vertica.log file or the SNMP trap utility to review the errors. For more information see Evaluating CRC Errors.</td>
</tr>
</tbody>
</table>
Event Data

To help you interpret and solve the issue that triggered an event, each event provides a variety of data, depending upon the event logging mechanism used.

The following table describes the event data and indicates where it is used.

<table>
<thead>
<tr>
<th>vertica.log</th>
<th>ACTIVE EVENTS (column names)</th>
<th>SNMP</th>
<th>Syslog</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>NODE_NAME</td>
<td>N/A</td>
<td>N/A</td>
<td>The node where the event occurred.</td>
</tr>
<tr>
<td>Event Code</td>
<td>EVENT_CODE</td>
<td>Event Type</td>
<td>Event Code</td>
<td>A numeric ID that indicates the type of event. See Event Types in the previous table for a list of event type codes.</td>
</tr>
<tr>
<td>Event Id</td>
<td>EVENT_ID</td>
<td>Event OID</td>
<td>Event Id</td>
<td>A unique numeric ID that identifies the specific event.</td>
</tr>
<tr>
<td>Event Severity</td>
<td>EVENT_SEVERITY</td>
<td>Event Severity</td>
<td>Event Severity</td>
<td>The severity of the event from highest to lowest. These</td>
</tr>
<tr>
<td>vertica.log</td>
<td>ACTIVE_EVENTS (column names)</td>
<td>SNMP</td>
<td>Syslog</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------</td>
<td>------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>events are based on standard syslog severity types: 0 – Emergency 1 – Alert 2 – Critical 3 – Error 4 – Warning 5 – Notice 6 – Info 7 – Debug</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostedTimestamp</td>
<td>EVENT_ POSTED_TIMESTAMP</td>
<td>N/A</td>
<td>PostedTimestamp</td>
<td>The year, month, day, and time the event was reported. Time is provided as military time.</td>
</tr>
<tr>
<td>ExpirationTimestamp</td>
<td>EVENT_EXPIRATION</td>
<td>N/A</td>
<td>ExpirationTimestamp</td>
<td>The time at which this event expires. If the same</td>
</tr>
<tr>
<td>vertica.log</td>
<td>ACTIVE_ EVENTS (column names)</td>
<td>SNMP</td>
<td>Syslog</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------</td>
<td>------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>event is posted again prior to its expiration time, this field gets updated to a new expiration time.</td>
</tr>
<tr>
<td>EventCodeDescription</td>
<td>EVENT_CODE_DESCRIPTION</td>
<td>Description</td>
<td>EventCodeDescription</td>
<td>A brief description of the event and details pertinent to the specific situation.</td>
</tr>
<tr>
<td>ProblemDescription</td>
<td>EVENT_PROBLEM_DESCRIPTION</td>
<td>Event Short Description</td>
<td>ProblemDescription</td>
<td>A generic description of the event.</td>
</tr>
<tr>
<td>N/A</td>
<td>REPORTING_NODE</td>
<td>Node Name</td>
<td>N/A</td>
<td>The name of the node within the cluster that reported the event.</td>
</tr>
<tr>
<td>DatabaseName</td>
<td>N/A</td>
<td>Database Name</td>
<td>DatabaseName</td>
<td>The name of the database that is impacted by the event.</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>Host Name</td>
<td>Hostname</td>
<td>The name of</td>
</tr>
</tbody>
</table>
The status of the event. It can be either:
1 – Open
2 – Clear

**Configuring Event Reporting**

Event reporting is automatically configured for `vertica.log`, and current events are automatically posted to the `ACTIVE_EVENTS` system table. You can also configure Vertica to post events to `syslog` and `SNMP`.

**Configuring Reporting for Syslog**

Syslog is a network-logging utility that issues, stores, and processes meaningful log messages. It is designed so DBAs can keep machines up and running, and it is a useful way to get heterogeneous data into a single data repository.

To log events to syslog, enable event reporting for each individual event you want logged. Messages are logged, by default, in `/var/log/messages`.

Configuring event reporting to syslog consists of:

1. Enabling Vertica to trap events for syslog.
2. Defining which events Vertica traps for syslog.
Vertica strongly suggests that you trap the Stale Checkpoint event.

3. Defining which syslog facility to use.

Enabling Vertica to Trap Events for Syslog

To enable event trapping for syslog, issue the following SQL command:

```sql
=> ALTER DATABASE mydb SET SyslogEnabled = 1;
```

To disable event trapping for syslog, issue the following SQL command:

```sql
=> ALTER DATABASE mydb SET SyslogEnabled = 0;
```

Defining Events to Trap for Syslog

To define events that generate a syslog entry, issue the following SQL command, where `Event_Name` is one of the events described in the list below the command:

```sql
=> ALTER DATABASE mydb SET SyslogEvents = 'Event_Name, Event_Name';
```

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Refresh Lock Error
- Tuple Mover Error
- Timer Service Task Error
- Stale Checkpoint

The following example generates a syslog entry for low disk space and recovery failure:

```sql
=> ALTER DATABASE mydb SET SyslogEvents = 'Low Disk Space, Recovery Failure';
```

Defining the SyslogFacility to Use for Reporting

The syslog mechanism allows for several different general classifications of logging messages, called facilities. Typically, all authentication-related messages are logged with the auth (or authpriv) facility. These messages are intended to be secure and hidden from unauthorized eyes. Normal operational messages are logged with the daemon facility, which is the collector that receives and optionally stores messages.

The SyslogFacility directive allows all logging messages to be directed to a different facility than the default. When the directive is used, all logging is done using the specified facility, both authentication (secure) and otherwise.

To define which SyslogFacility Vertica uses, issue the following SQL command:

```sql
=> ALTER DATABASE mydb SET SyslogFacility = 'Facility_Name';
```

Where the facility-level argument `<Facility_Name>` is one of the following:

<table>
<thead>
<tr>
<th>Facility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td></td>
</tr>
<tr>
<td>authpriv (Linux only)</td>
<td>local0 (local use 0)</td>
</tr>
<tr>
<td>cron</td>
<td>local1 (local use 1)</td>
</tr>
<tr>
<td>daemon</td>
<td>local2 (local use 2)</td>
</tr>
<tr>
<td>ftp (Linux only)</td>
<td>local3 (local use 3)</td>
</tr>
<tr>
<td>lpr (line printer subsystem)</td>
<td>local4 (local use 4)</td>
</tr>
<tr>
<td>mail (mail system)</td>
<td>local5 (local use 5)</td>
</tr>
</tbody>
</table>
### Configuring Reporting for SNMP

Configuring event reporting for SNMP consists of:

1. **Configuring Vertica to enable event trapping for SNMP** as described below.

2. **Importing the Vertica Management Information Base (MIB) file** into the SNMP monitoring device.

   The Vertica MIB file allows the SNMP trap receiver to understand the traps it receives from Vertica. This, in turn, allows you to configure the actions it takes when it receives traps.

   Vertica supports the SNMP V1 trap protocol, and it is located in `/opt/vertica/sbin/VERTICA-MIB`. See the documentation for your SNMP monitoring device for more information about importing MIB files.

3. **Configuring the SNMP trap receiver to handle traps from Vertica.**

   SNMP trap receiver configuration differs greatly from vendor to vendor. As such, the directions presented here for configuring the SNMP trap receiver to handle traps from Vertica are generic.

   Vertica traps are single, generic traps that contain several fields of identifying information. These fields equate to the event data described in [Monitoring Events](#). However, the format used for the field names differs slightly. Under SNMP, the field names contain no spaces. Also, field names are pre-pended with “vert”. For example, Event Severity becomes `vertEventSeverity`.

   When configuring your trap receiver, be sure to use the same hostname, port, and community string you used to configure event trapping in Vertica.
Examples of network management providers:

- Network Node Manager
- IBM Tivoli
- AdventNet
- Net-SNMP (Open Source)
- Nagios (Open Source)
- Open NMS (Open Source)

See Also

- Configuration Parameters

Configuring Event Trapping for SNMP

The following events are trapped by default when you configure Vertica to trap events for SNMP:

- Low Disk Space
- Read Only File System
- Loss of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Stale Checkpoint
- CRC Mismatch
To Configure Vertica to Trap Events for SNMP

1. Enable Vertica to trap events for SNMP.
2. Define where Vertica sends the traps.
3. Optionally redefine which SNMP events Vertica traps.

**Note:** After you complete steps 1 and 2 above, Vertica automatically traps the default SNMP events. Only perform step 3 if you want to redefine which SNMP events are trapped. Vertica recommends that you trap the Stale Checkpoint event even if you decide to reduce the number of events Vertica traps for SNMP. The specific settings you define have no effect on traps sent to the log. All events are trapped to the log.

To Enable Event Trapping for SNMP

Use the following SQL command:

```
=> ALTER DATABASE mydb SET SnmpTrapsEnabled = 1;
```

To Define Where Vertica Send Traps

Use the following SQL command, where Host_name and port identify the computer where SNMP resides, and CommunityString acts like a password to control Vertica’s access to the server:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'host_name port CommunityString';
```

For example:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'localhost 162 public';
```

You can also specify multiple destinations by specifying a list of destinations, separated by commas:

```
=> ALTER DATABASE mydb SET SnmpTrapDestinationsList = 'host_name1 port1 CommunityString1, hostname2 port2 CommunityString2';
```

**Note:** Setting multiple destinations sends any SNMP trap notification to all destinations listed.
To Define Which Events Vertica Traps

Use the following SQL command, where Event_Name is one of the events in the list below the command:

```sql
=> ALTER DATABASE mydb SET SnmpTrapEvents = 'Event_Name1, Event_Name2';
```

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Tuple Mover Error
- Stale Checkpoint
- CRC Mismatch

**Note:** The above values are case sensitive.

The following example specifies two event names:

```sql
=> ALTER DATABASE mydb SET SnmpTrapEvents = 'Low Disk Space, Recovery Failure';
```
Verifying SNMP Configuration

To create a set of test events that checks SNMP configuration:

1. Set up SNMP trap handlers to catch Vertica events.
2. Test your setup with the following command:

```
SELECT SNMP_TRAP_TEST();
SNMP_TRAP_TEST
---------------------
Completed SNMP Trap Test
(1 row)
```

Event Reporting Examples

Vertica.log

The following example illustrates a Too Many ROS Containers event posted and cleared within vertica.log:

```
08/14/15 15:08:54 thr:Ageout Events:0x2aaab0015e70 [INFO] Event Cleared:
```

SNMP

The following example illustrates a Too Many ROS Containers event posted to SNMP:

```
Version: 1, type: TRAPREQUESTEnterprise OID: .1.3.6.1.4.1.31207.2.0.1
Trap agent: 72.0.0.0
Generic trap: ENTERPRISESPECIFIC (6)
Specific trap: 0
.1.3.6.1.4.1.31207.1.1 --> 4
```
Syslog

The following example illustrates a Too Many ROS Containers event posted and cleared within syslog:


Using System Tables

Vertica provides an API (application programming interface) for monitoring various features and functions within a database in the form of system tables. These tables provide a robust, stable set of views that let you monitor information about your system's resources, background processes, workload, and performance, allowing you to more efficiently profile, diagnose, and view historical data equivalent to load streams, query profiles, tuple mover operations, and more. Because Vertica collects and retains this information automatically, you don't have to manually set anything.

You can write queries against system tables with full SELECT support the same way you perform query operations on base and temporary tables. You can query system tables using expressions, predicates, aggregates, analytics, subqueries, and joins. You can also save system table query results into a user table for future analysis. For example, the following query creates a table, mynode, selecting three node-related columns from the V_CATALOG.NODES system table:

```
VMart=> CREATE TABLE mynode AS SELECT node_name, node_state, node_address
FROM nodes;
VMart=> CREATE TABLE
VMart=> SELECT * FROM mynode;

node_name | node_state | node_address
------------------------+-------------+-----------------+
v_vmart_node0001 | UP         | 192.168.223.11
(1 row)
```

Note: You cannot query system tables if the database cluster is in a recovering state. The database refuses connection requests and cannot be monitored. Vertica also does not support DDL and DML operations on system tables.

Where System Tables Reside

System tables are grouped into the following schemas:

- **V_CATALOG** — information about persistent objects in the catalog
- **V_MONITOR** — information about transient system state

These schemas reside in the default search path so there is no need to specify schema.table in your queries unless you change the search path to exclude V_MONITOR or V_CATALOG or both.
The system tables that make up the monitoring API are described fully in the [SQL Reference Manual](#). You can also use the following command to view all the system tables and their schema:

```sql
SELECT * FROM system_tables ORDER BY table_schema, table_name;
```

## How System Tables Are Organized

Most of the tables are grouped into the following areas:

- System information
- System resources
- Background processes
- Workload and performance

Vertica reserves some memory to help monitor busy systems. Using simple system table queries makes it easier to troubleshoot issues. See also SYSQUERY and SYSDATA pools under the Built-in pools topic in the SQL Reference Manual.

**Note:** You can use external monitoring tools or scripts to query the system tables and act upon the information, as necessary. For example, when a host failure causes the K-safety level to fall below the desired level, the tool or script can notify the database administrator and/or appropriate IT personnel of the change, typically in the form of an e-mail.

## Privileges

You can GRANT and REVOKE privileges on system tables, with the following restrictions:

- You cannot GRANT privileges on system tables to the SYSMONITOR or PSEUDOSUPERUSER roles.
- You cannot GRANT on system schemas.

## Querying Case-Sensitive Data in System Tables

Some system table data might be stored in mixed case. For example, Vertica stores mixed-case identifier names the way you specify them in the CREATE statement, even though case is
ignored when you reference them in queries. When these object names appear as data in the system tables, you'll encounter errors if you use the equality (\(=\)) predicate because the case must match the stored identifier. In particular, V\_CATALOG\_TABLE\_SCHEMA and V\_CATALOG\_TABLE\_NAME columns are case sensitive with equality predicates.

If you don't know how the identifiers are stored, use the case-insensitive operator \(\text{ILIKE}\) instead of equality predicates.

For example, given the following schema:

```sql
-> CREATE SCHEMA SS;
-> CREATE TABLE SS.TT (c1 int);
-> CREATE PROJECTION SS.TTP1 AS SELECT * FROM ss.tt UNSEGMENTED ALL NODES;
-> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the \(=\) predicate, Vertica returns 0 rows:

```sql
-> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema = 'ss';
<table>
<thead>
<tr>
<th>table_schema</th>
<th>table_name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(0 rows)
```

Using the case-insensitive \(\text{ILIKE}\) predicate returns the expected results:

```sql
-> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'ss';
<table>
<thead>
<tr>
<th>table_schema</th>
<th>table_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>TT</td>
</tr>
</tbody>
</table>
(1 row)
```

**Examples**

The following query uses the VMart example database (see *Introducing the VMart Example Database*) to obtain the number of rows and size occupied by each table in the database.

```sql
-> SELECT t.table_name AS table_name,
     SUM(ps.wos_row_count + ps.ros_row_count) AS row_count,
     SUM(ps.wos_used_bytes + ps.ros_used_bytes) AS byte_count
FROM tables t
JOIN projections p ON t.table_id = p.anchor_table_id
JOIN projection_storage ps ON p.projection_name = ps.projection_name
WHERE (ps.wos_used_bytes + ps.ros_used_bytes) > 500000
GROUP BY t.table_name
ORDER BY byte_count DESC;
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>row_count</th>
<th>byte_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>online_sales_fact</td>
<td>5000000</td>
<td>171987438</td>
</tr>
<tr>
<td>store_sales_fact</td>
<td>5000000</td>
<td>108844666</td>
</tr>
<tr>
<td>store_orders_fact</td>
<td>300000</td>
<td>9240800</td>
</tr>
<tr>
<td>product_dimension</td>
<td>60000</td>
<td>2327964</td>
</tr>
<tr>
<td>customer_dimension</td>
<td>50000</td>
<td>2165897</td>
</tr>
</tbody>
</table>
The rest of the examples illustrate simple ways to use system tables in queries.

```sql
=> SELECT table_name FROM columns WHERE data_type ILIKE 'Numeric' GROUP BY table_name;
table_name
----------
n1
(1 row)
=> SELECT current_epoch, designed_fault_tolerance, current_fault_tolerance FROM SYSTEM;
current_epoch | designed_fault_tolerance | current_fault_tolerance
---------------|--------------------------|--------------------------
492           | 1                        | 1
(1 row)
=> SELECT node_name, total_user_session_count, executed_query_count FROM query_metrics;
node_name | total_user_session_count | executed_query_count
----------|--------------------------|--------------------------
v_vmart_node0001 | 53 | 42
v_vmart_node0002 | 53 | 0
v_vmart_node0003 | 42 | 120
v_vmart_node0004 | 53 | 0
(4 rows)
=> SELECT table_schema FROM primary_keys;
table_schema
----------
public
public
public
public
public
public
public
public
store
online_sales
online_sales
(12 rows)
```
Retaining Monitoring Information

When you query a Vertica system table (described in Using System Tables), you can get information about currently running queries, the state of various components, and other runtime information. During query execution, Vertica examines the current state of the system and returns information in the result set.

Data Collector

Vertica also provides a utility called the Data Collector (DC), which collects and retains history of important system activities and records essential performance and resource utilization counters.

Data Collector extends system table functionality by:

- Providing a framework for recording events
- Making the information available in system tables
- Requiring few configuration parameter tweaks
- Having negligible impact on performance

You can use the information the Data Collector retains to query the past state of system tables and extract aggregate information, as well as do the following:

- See what actions users have taken
- Locate performance bottlenecks
- Identify potential improvements to Vertica configuration

DC does not collect data for nodes that are down, so no historical data would be available for that node.

Data Collector works in conjunction with the Workload Analyzer (WLA), an advisor tool that intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history. See Analyzing Workloads for more information about WLA.
Where Is DC Information retained?

Collected data is stored on disk in the DataCollector directory under the Vertica /catalog path. This directory also contains instructions on how to load the monitoring data into another Vertica database. See Working with Data Collection Logs for details.

DC retains the data it gathers based on retention policies, which a superuser can configure. See Configuring Data Retention Policies.

Data Collector is on by default, but a superuser can disable it if performance issues arise. See Data Collector Parameters and Enabling and Disabling Data Collector.

DC Tables

Caution: Data Collector tables (prefixed by dc_) reside in the V_INTERNAL schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after a Vertica upgrade. [[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you use the Workload Analyzer instead of accessing the Data Collector tables directly.

See Also

- Data Collector Functions
- DATA_COLLECTOR
- ANALYZE_WORKLOAD
- TUNING_RECOMMENDATIONS

Enabling and Disabling Data Collector

Data Collector is on by default and retains information for all sessions. If performance issues arise, a superuser can disable Data Collector at any time.

To disable the Data Collector:
=> ALTER DATABASE mydb SET EnableDataCollector = 0;

To re-enable the Data Collector:

=> ALTER DATABASE mydb SET EnableDataCollector = 1;

See Also

- Data Collector Parameters

Viewing Current Data Retention Policy

To view the current retention policy for a Data Collector component, use the `GET_DATA_COLLECTOR_POLICY()` function and supply the component name as the function's argument.

To retrieve a list of all current component names, query the `V_MONITOR.DATA_COLLECTOR` system table, which returns Data Collector components, their current retention policies, and statistics about how much data is retained. For example:

```
mcdb=> \xExpanded display is on.
mcdb=> SELECT * from data_collector;
-[ RECORD 1 ]-+-----------------------------------------------+----------------+
          | node_name         | v_mcdb_node0001  |
          | component         | AllocationPoolStatistics |
          | table_name         | dc_allocation_pool_statistics  |
          | description         | Information about global memory pools, which generally cannot be recovered |
          | without restart     | t                  |
          | access_restricted   | f                  |
          | in_db_log           | f                  |
          | in_vertica_log      | f                  |
          | memory_buffer_size_kb | 64                 |
          | disk_size_kb        | 256                |
          | record_too_big_errors | 0                  |
          | lost_buffers        | 0                  |
          | lost_records        | 0                  |
          | retired_files       | 1429               |
          | retired_records     | 647358             |
          | current_memory_records | 0                  |
          | current_disk_records | 1493               |
          | current_memory_bytes | 0                  |
          | current_disk_bytes  | 215737             |
          | first_time          | 2012-11-30 07:04:30.000726-05 |
          | last_time           | 2012-11-30 07:16:56.000631-05 |
          | kb_per_day          | 24377.3198211312   |
-| [ RECORD 2 ]-+-----------------------------------------------+----------------+---
          | node_name         |                     |
```

The following command returns the retention policy for a specific component, `NodeState`. 

```
The results let you know that 10KB is retained in memory and 100KB on disk:

```
get_data_collector_policy-------------------------------
10KB kept in memory, 100KB kept on disk.
(1 row)
```

### Configuring Data Retention Policies

Data Collector retention policies hold the following information:

- Which component to monitor
- How much memory to retain
- How much disk space to retain

A superuser can modify policies, such as change the amount of data to retain, by invoking the `SET_DATA_COLLECTOR_POLICY()` function, as follows:

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB')
```

The `SET_DATA_COLLECTOR_POLICY()` function sets the retention policy for the specified component on all nodes, and lets you specify memory and disk size to retain in kilobytes. Failed nodes receive the new setting when they rejoin the cluster.

For example, the following statement specifies that the NodeState component be allocated 50KB of memory and 250KB of disk space:

```
=> SELECT SET_DATA_COLLECTOR_POLICY('NodeState', '50', '250');
SET_DATA_COLLECTOR_POLICY-------------------------------
SET
(1 row)
```

Before you change a retention policy, you can view the current setting by calling the `GET_DATA_COLLECTOR_POLICY()` function.

You can also use the `GET_DATA_COLLECTOR_POLICY()` function to verify changed settings. For example, the following query retrieves a brief statement about the retention policy for the NodeState component:

```
=> SELECT GET_DATA_COLLECTOR_POLICY('NodeState');
GET_DATA_COLLECTOR_POLICY-------------------------------
50KB kept in memory, 250KB kept on disk.
(1 row)
```
Tip: If you do not know the name of a component, you can query the `V_MONITOR.DATA_COLLECTOR` system table to get a full list. For example, the following query returns all current Data Collector components and a description of each:

```sql
> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

See Also

- `GET_DATA_COLLECTOR_POLICY`
- `SET_DATA_COLLECTOR_POLICY`

Working with Data Collection Logs

Upon startup, a Vertica database creates a DataCollector directory within the `/catalog` directory.

The DataCollector directory holds the disk-based data collection logs, where retained data is kept in files named `<component>_<timestamp>.log`. Vertica might maintain several log files, per component, at any given time. See Querying Data Collector Tables for an example of how to view this information.

Also upon startup, Vertica creates two additional files, per component, in the DataCollector directory. These are SQL files that contain examples on how to load Data Collector data into another Vertica instance. These files are:

- `CREATE_<component>_TABLE.sql` — contains the SQL DDL needed to create a table into which Data Collector logs for the component can be loaded.

- `COPY_<component>_TABLE.sql` — contains example SQL to load (using `COPY` commands) the data log files into the table that the `CREATE` script creates.

Two functions let you manipulate these log files.

<table>
<thead>
<tr>
<th>If you want to ...</th>
<th>See these topics in the SQL Reference Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear memory and disk records from Data Collector retention and reset collection statistics</td>
<td><code>CLEAR_DATA_COLLECTOR()</code></td>
</tr>
<tr>
<td>If you want to ...</td>
<td>See these topics in the SQL Reference Manual</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>Flush the Data Collector logs</td>
<td>FLUSH_DATA_COLLECTOR()</td>
</tr>
<tr>
<td>Retrieve a list of all current Data Collector components</td>
<td>V_MONITOR.DATA_COLLECTOR</td>
</tr>
</tbody>
</table>

### Clearing the Data Collector

If you want to clear the Data Collector of all memory and disk records and reset the collection statistics in the V_MONITOR.DATA_COLLECTOR system table, call the CLEAR_DATA_COLLECTOR() function. You can clear records for all components on all nodes or you can specify individual components, one at a time.

To clear records on a single component, pass the function the component argument. For example, the following command clears records for the ResourceAcquisitions component only, returning a result of CLEAR (success):

```
=> SELECT clear_data_collector('ResourceAcquisitions');
clear_data_collector
------------------------
CLEAR
(1 row)
```

The following command clears the Data Collector records for all components on all nodes:

```
=> SELECT clear_data_collector();
clear_data_collector
------------------------
CLEAR
(1 row)
```

**Note:** After you clear the DataCollector log, the information is no longer available for querying.

### Flushing Data Collector Logs

If you want to flush Data Collector information for all components or for an individual component, use the FLUSH_DATA_COLLECTOR() function. This function waits until memory logs are moved to disk and flushes the Data Collector, synchronizing the log with the disk storage:

To flush data collection for all components on all nodes:
To flush records on a single component, pass a component argument to the function. For example, the following command flushes the ResourceAcquisitions component:

```
=> SELECT flush_data_collector('ResourceAcquisitions');
flush_data_collector
--------------------------
FLUSH
(1 row)
```

See Also

- Data Collector Functions
- DATA_COLLECTOR

Monitoring Data Collection Components

Query the V_MONITOR.DATA_COLLECTOR system table to get a list of Data Collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. DATA_COLLECTOR also calculates the approximate collection rate, to aid in sizing calculations.

The following is a simple query that returns all the columns in this system table. See V_MONITOR.DATA_COLLECTOR in the SQL Reference Manual for additional details.

```
=> \Expanddisplay is on.
=> SELECT * FROM data_collector;
-[ RECORD 1 ]--------------------------------------------
node_name | v_vmartdb_node001
component | AllocationPoolStatistics
table_name | dc_allocation_pool_statistics
description | Information about global memory pools, ...
in_db_log | f
in_vertica_log | f
memory_buffer_size_kb | 64
disk_size_kb | 256
record_too_big_errors | 0
lost_buffers | 0
lost_records | 0
retired_files | 120
retired_records | 53196
current_memory_records | 0
```
Related Topics

V_MONITOR.DATA_COLLECTOR and Data Collector Functions in the SQL Reference Manual
Retaining Monitoring Information and How Vertica Calculates Database Size in this guide

Querying Data Collector Tables

Caution: Data Collector tables (prefixed by dc_) reside in the V_INTERNAL schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after a Vertica upgrade.

[[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you use the Workload Analyzer instead of accessing the Data Collector tables directly.

Here's one useful example you can use to check on the state of your database. Upon startup, the Vertica database creates, under its catalog directory, a DataCollector directory. This directory holds the disk-based data collection logs. The main data is kept in files named <component>.<timestamp>.log.

When you start your database an entry is created in the dc_startups table. The following is the result of querying this table.

```sql
=> SELECT * FROM dc_startups;
-[ RECORD 1 ]---------------------------------------------------------------------------------------------------
time | 2011-05-26 17:35:40.588589-04
node_name | v_vmartdb_node0001
version | Vertica Analytic Database v5.0.4-20110526
command_line | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/catalog/vmartdb/v vmartdb_node0001_catalog -h 10.10.50.123 -p 5608
codename | 5.0
build_tag | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu May 26 3:37:18 2011' $BuildId$
build_type | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale | UTF-8
```
Monitoring Query Plan Profiles

See Profiling Query Plans.

Monitoring Partition Reorganization

When you use `ALTER TABLE ... REORGANIZE`, the operation reorganizes the data in the background.

You can monitor details of the reorganization process by polling the following system tables:
- **V_MONITOR.PARTITION_STATUS** displays the fraction of each table that is partitioned correctly.

- **V_MONITOR.PARTITION_REORGANIZE_ERRORS** logs errors issued by the reorganize process.

- **V_MONITOR.PARTITIONS** displays NULL in the partition_key column for any ROS that was not reorganized.

Note: The corresponding foreground process to `ALTER TABLE...REORGANIZE` is `PARTITION TABLE`.

### Using Queries to Monitor Resource Pool Size and Usage

You can use the Linux **top command** to determine the overall CPU usage and I/O wait time across the system. However, because of file system caching, the resident memory size indicated by `top` is not a good indicator of actual memory use or available reserve.

Instead, Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests, and the state of the queues.

The **RESOURCE_POOLS** table lets you view various resource pools defined in the system (both internal and user-defined), and the **RESOURCE_POOL_STATUS** table lets you view the current state of the resource pools.

### Examples

This example shows how to find the various resource pools defined in the system.

```sql
=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;

<table>
<thead>
<tr>
<th>name</th>
<th>memorysize</th>
<th>maxmemorysize</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td></td>
<td>Special: 95%</td>
</tr>
<tr>
<td>sysquery</td>
<td>64M</td>
<td></td>
</tr>
<tr>
<td>sysdata</td>
<td>100M</td>
<td>10%</td>
</tr>
<tr>
<td>wosdata</td>
<td>0%</td>
<td>25%</td>
</tr>
<tr>
<td>tm</td>
<td>200M</td>
<td></td>
</tr>
<tr>
<td>refresh</td>
<td>0%</td>
<td></td>
</tr>
</tbody>
</table>
```
Viewing Only User-Defined Resource Pools

To see only the user-defined resource pools, you can limit your query to return records where `IS_INTERNAL` is false.

Note: The user-defined pools shown in these examples also appear in subsequent sections related to Workload Management.

This example shows how to find information on user-defined resource pools:

=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency
   FROM V_CATALOG RESOURCE_POOLS where is_internal = 'f';

<table>
<thead>
<tr>
<th>name</th>
<th>memorysize</th>
<th>maxmemorysize</th>
<th>priority</th>
<th>maxconcurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_pool</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ceo_pool</td>
<td>250M</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>ad_hoc_pool</td>
<td>200M</td>
<td>200M</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>billing_pool</td>
<td>0%</td>
<td></td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>web_pool</td>
<td>25M</td>
<td></td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>batch_pool</td>
<td>150M</td>
<td>150M</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>dept1_pool</td>
<td>0%</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>dept2_pool</td>
<td>0%</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

(8 rows)

Viewing the Status of All Resource Pools

The following example shows how to access the V_MONITOR RESOURCE_POOL_STATUS table to return the current state of all resource pools on node0001:

=> \x
Expanded display is on

=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb, general_memory_borrowed_kb, running_query_count
   FROM V_MONITOR RESOURCE_POOL_STATUS where node_name ilike '%node0001';

- [ RECORD 1 ]-----------------
  pool_name | general
<table>
<thead>
<tr>
<th>pool_name</th>
<th>sysquery</th>
<th>sysdata</th>
<th>wosdata</th>
<th>tm</th>
<th>refresh</th>
<th>recovery</th>
<th>dbd</th>
<th>jvm</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_size_kb</td>
<td>2983177</td>
<td>65536</td>
<td>0</td>
<td>204800</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>memory_size_actual_kb</td>
<td>2983177</td>
<td>65536</td>
<td>0</td>
<td>204800</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>memory_inuse_kb</td>
<td>0</td>
<td>4096</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>general_memory_borrowed_kb</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>running_query_count</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- [ RECORD 2 ]----------
- [ RECORD 3 ]----------
- [ RECORD 4 ]----------
- [ RECORD 5 ]----------
- [ RECORD 6 ]----------
- [ RECORD 7 ]----------
- [ RECORD 8 ]----------
- [ RECORD 9 ]----------
Viewing Query Resource Acquisitions

The following example displays all resources granted to the queries that are currently running. The information shown is stored in the  V_MONITOR.RESOURCE_ACQUISITIONS table. You can see that the query execution used 708504 KB of memory from the GENERAL pool.

```sql
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
    queue_entry_timestamp, acquisition_timestamp
    FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

<table>
<thead>
<tr>
<th>pool_name</th>
<th>thread_count</th>
<th>open_file_handle_count</th>
<th>memory_inuse_kb</th>
<th>queue_entry_timestamp</th>
<th>acquisition_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysquery</td>
<td>4</td>
<td>0</td>
<td>4108</td>
<td>2013-12-05 07:07:08.815362-05</td>
<td>2013-12-05 07:07:08.815367-05</td>
</tr>
</tbody>
</table>

You can determine how long a query waits in the queue before it can run. To do so, you obtain the difference between the acquisition_timestamp and the queue_entry_timestamp using a query as this example shows:

```sql
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
    (acquisition_timestamp - queue_entry_timestamp) AS 'queue wait'
    FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

<table>
<thead>
<tr>
<th>pool_name</th>
<th>queue_entry_timestamp</th>
<th>acquisition_timestamp</th>
<th>queue wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysquery</td>
<td>2013-12-05 07:07:08.815362-05</td>
<td>2013-12-05 07:07:08.815367-05</td>
<td>00:00:00.000005</td>
</tr>
<tr>
<td>sysquery</td>
<td>2013-12-05 07:07:14.714412-05</td>
<td>2013-12-05 07:07:14.714417-05</td>
<td>00:00:00.000005</td>
</tr>
<tr>
<td>sysquery</td>
<td>2013-12-05 07:09:57.238521-05</td>
<td>2013-12-05 07:09:57.238521-05</td>
<td>00:00:00.000005</td>
</tr>
</tbody>
</table>
See Also

- See the SQL Reference Manual for detailed descriptions of the monitoring tables.
- See Monitoring Resource Pools for descriptions of other ways to monitor resource usage.

Monitoring Recovery

When your Vertica database is recovering from a failure, it's important to monitor the recovery process. There are several ways to monitor database recovery:

Viewing Log Files on Each Node

During database recovery, Vertica adds logging information to the vertica.log on each host. Each message is identified with a [Recover] string.

Use the tail command to monitor recovery progress by viewing the relevant status messages, as follows.

```bash
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing host v_vmart_node0001 startup state from INITIALIZING to RECOVERING
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to specified epoch 0x120b6
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split queries
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query: ALTER PROJECTION proj_tradesquotes_0 SPLIT v_vmart_node0001 FROM 73911;
```

Using System Tables to Monitor Recovery

Use the following system tables to monitor recover:
RECOVERY_STATUS

PROJECTION_RECOVERIES

Specifically, the recovery_status system table includes information about the node that is recovering, the epoch being recovered, the current recovery phase, and running status:

```sql
=> select node_name, recover_epoch, recovery_phase, current_completed, is_running from recovery_status;
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>recover_epoch</th>
<th>recovery_phase</th>
<th>current_completed</th>
<th>is_running</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td></td>
<td></td>
<td>0</td>
<td>f</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>0</td>
<td>historical pass 1</td>
<td>0</td>
<td>t</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>1</td>
<td>current</td>
<td>0</td>
<td>f</td>
</tr>
</tbody>
</table>

The projection_recoveries system table maintains history of projection recoveries. To check the recovery status, you can summarize the data for the recovering node, and run the same query several times to see if the counts change. Differing counts indicate that the recovery is working and in the process of recovering all missing data:

```sql
=> select node_name, status, progress from projection_recoveries;
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>status</th>
<th>progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>running</td>
<td>61</td>
</tr>
</tbody>
</table>

To see a single record from the projection_recoveries system table, add limit 1 to the query.

After a recovery has completed, Vertica continues to store information from the most recent recovery in these tables.

Viewing Cluster State and Recovery Status

Use the admintools view_cluster tool from the command line to see the cluster state:

```bash
$ /opt/vertica/bin/admintools -t view_cluster
```

<table>
<thead>
<tr>
<th>DB</th>
<th>Host</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>112.17.31.10</td>
<td>RECOVERING</td>
</tr>
<tr>
<td></td>
<td>112.17.31.11</td>
<td>UP</td>
</tr>
<tr>
<td>&lt;data_base&gt;</td>
<td>112.17.31.12</td>
<td>UP</td>
</tr>
<tr>
<td>&lt;data_base&gt;</td>
<td>112.17.31.17</td>
<td>UP</td>
</tr>
</tbody>
</table>

Monitoring Cluster Status After Recovery

When recovery has completed:
1. Launch Administration Tools.

2. From the Main Menu, select View Database Cluster State and click OK.

   The utility reports your node's status as UP.

   **Note:** You can also monitor the state of your database nodes on the Management Console Overview page under the Database section, which tells you the number of nodes that are up, critical, recovering, or down. To get node-specific information, click Manage at the bottom of the page.

### Clearing Projection Refresh History

System table `PROJECTION_REFRESHES` records information about refresh operations, successful and unsuccessful. `PROJECTION_REFRESHES` retains refresh data until one of the following events occurs:

- `CLEAR_PROJECTION_REFRESHES` is called.
- The table's storage quota is exceeded.

To immediately purge this information, call `CLEAR_PROJECTION_REFRESHES`:

```
=> SELECT clear_projection_refreshes();

    clear_projection_refreshes
     -------------------------
       CLEAR
          (1 row)
```

**Note:** `PROJECTION_REFRESHES` checks the Boolean column `IS_EXECUTING` in `PROJECTION_REFRESHES` to determine whether refresh operations are still running or are complete. The function only removes information for refresh operations that are complete.

### See Also

`PROJECTION_REFRESHES`
**Monitoring Vertica Using Notifiers**

A Vertica notifier is a push-based mechanism capable of sending messages from Vertica to endpoints such as Apache Kafka. For example, you could configure a long-running script to send notifications at various stages and then at the completion of a task.

To use a notifier, follow these steps:

1. Create a notifier with `CREATE NOTIFIER`.
2. Use the notifier to send messages from Vertica to the endpoint with the meta-function `NOTIFY`.

**See Also**

`ALTER NOTIFIER`
Back Up and Restoring the Database

Creating regular database backups is an important part of basic maintenance tasks at your site. Vertica supplies a comprehensive utility, called vbr, for this purpose. This utility lets you back up, restore, list backups, and copy your database to another cluster. You can create full and incremental database backups, and backups of schemas or tables (object-level backups) for use with a multi-tenanted database. When a full backup exists, you can restore one or more objects from the backup or the entire database, as required.

Using vbr, you can save your data to a variety of locations:

- A local directory on the nodes in the cluster
- One or more hosts outside of the cluster
- A different Vertica cluster (effectively cloning your database)
- Amazon S3 storage

Impact of Backups on Vertica Nodes

While a backup is taking place, the backup process can consume additional storage. The amount of space consumed depends on the size of your catalog and any objects that you drop during the backup. The backup process releases this storage once the backup is complete.

Compatibility Requirements for Restore and Replication

Creating backups with vbr requires restoring backups with the same utility. Vertica 6.x and later supports object-level backups.

Vertica supports restore, replication, and copycluster actions only to the same exact version of Vertica that created the backup. For example, you cannot restore, replicate objects, or copycluster a version 7.0.2-3 backup to a version 7.0.2-5 database.
Additional Considerations for HDFS Storage Locations

If your database has any storage locations on HDFS, you must do additional configuration to enable those storage locations to be backed up. See Backing Up HDFS Storage Locations in Integrating with Apache Hadoop.

Supported File Systems

Vertica supports the following file systems for backup and temporary directory locations:

- ext3
- ext4
- NFS
- Amazon S3 Standard

The file system at your backup and temporary directory locations must support fcntl lockf (POSIX) file locking. Backups to Amazon S3 use a secondary location to handle file locking.
Creating vbr Configuration Files

The vbr utility uses a configuration file for the information required to back up and restore a full- or object-level backup or copy a cluster. You cannot run vbr without a configuration file because no default file exists. You can, however:

- Create a configuration file. See Start the vbr Configuration Script.
- Copy and edit an existing configuration file and save the changes to a different file name.
- Copy and edit one of the sample configuration files included with Vertica.

**Note:** You must be logged on as the database administrator, not root, to create the vbr configuration file.

Start the vbr Configuration Script

To create a configuration file, enter this command:

```bash
> /opt/vertica/bin/vbr --setupconfig
```

The script prompts you to answer several questions, as shown in the following summarized example:

```
[dbadmin@localhost ]$ /opt/vertica/bin/vbr --setupconfig
Snapshot name (backup_snapshot): fullbak1
Number of restore points (1): 3
Specify objects (no default):
Object restore mode (coexist, createOrReplace or create) (createOrReplace):
Vertica user name (dbadmin):
Save password to avoid runtime prompt? (n) [y/n]: y
Password to save in vbr config file (no default):
Node v_vmart_node0001
Backup host name (no default): 194.66.82.11
Backup directory (no default): /home/dbadmin/backups
Config file name (fullbak1.ini):
Password file name (no default value) (no default): pwdfile
Change advanced settings? (n) [y/n]: n
Saved vbr configuration to fullbak1.ini.
Saved vbr database password to pwdfile.ini.
```

For further information on how to answer these questions, see:

See the following instructions to complete the questions in the configuration script:
Specify a Backup Name

Specify Restore Points

Specify Object Restore Mode

Specify Full or Object-Level Backups

Enter the User Name

Save the Account Password

Specify the Backup Host and Directory

Save the Configuration File

Continue to Advanced Settings

Specify a Backup Name

Specify the name of the backup for vbr to create:

```
Snapshot name (backup_snapshot):
```

You must provide a name for the backup that clearly identifies the specific backup. The backup name is used in the directory tree structure that vbr creates for each node. Because Vertica automatically augments the backup subdirectories with date and time indicators, do not add dates to the backup name.

Create different configuration files with specific backup names for full and object-level backups, but use the same backup directory for both types of backups. For example, the configuration file for a full database backup, namedfullbak.ini, has these snapshotName and backupDir parameter values:

```
snapshotName=fullbak
backupDir=/home/dbadmin/data/backups
```

The configuration file for the object-level backup, namedobjectbak.ini, has these parameter values:

```
snapshotName=objectbak
backupDir=/home/dbadmin/data/backups
```
Specify Restore Points

Specify how many backups to save as a number of restore points:

Number of restore points (1):

The default value is 1, indicating that vbr always retains one additional restore point.

Saving multiple restore points gives you the option to recover from one of several backups. For example, if you specify 3, you have 1 current backup, and 3 backup archives. Vertica stores the value you enter as the restorePointLimit parameter in the vbr configuration file.

Specify Object Restore Mode

Specify how Vertica should handle restored objects, as this example shows:

Object restore mode (coexist, createOrReplace or create) (createOrReplace):

Vertica supports the following object restore modes:

- createOrReplace (default) — Vertica creates any objects that do not exist. If the object does exist, vbr overwrites it with the version from the archive.

- create — Vertica creates any objects that do not exist. If an object being restored does exist, Vertica displays an error message and skips that object.

- coexist — Vertica creates any objects that do not exist. If the object does exist, Vertica creates the restored version with a name of the form <backup>_<timestamp>_<object_name>. This approach allows existing and restored objects to exist simultaneously. If the appended information pushes the schema name past the maximum length of 128 characters, HP Vertica truncates the name. You can perform a reverse lookup of the original schema name by querying the system table TRUNCATED_SCHEMA.

In all modes, Vertica restores data with the current epoch. Object restore mode settings do not apply to backups and full restores.

Specify Full or Object-Level Backups

Indicate whether you want to create a full- or object-level backup:
Specify objects (no default):

- For full database backup: Press Enter to continue creating a configuration file for a full database backup.

- For object-level backup: Enter the names of any schema or table for which you want to create an object-level backup configuration file. Enter table names in the form `schema.objectname`.

  For example, to make backups of the table, `customers` from the schema, `finance`, enter `finance.customers`. Separate each name with a comma (,). The objects you enter appear listed in the Objects parameter of the configuration file.

Enter the User Name

Enter the user name of the person who invoking `vbr`:

Vertica user name (dbadmin):

Note: This user name must be the user identified as the database administrator for the entire database. No other user has permission to invoke this action, even if the user has dbadmin permissions within portions of the database.

Save the Account Password

Specify whether `vbr` prompts for an account password at run time:

Save password to avoid runtime prompt? (n) [y/n]:

Press Enter to accept the default (n) and continue. Because you are not saving the password, you must enter it when you run `vbr`.

To save your password to a password configuration file, enter y. The utility prompts for you to enter the password but does not display the text as you type:

Save password to avoid runtime prompt? (n) [y/n]:

The parameter name in the configuration file that indicates whether to prompt for a password is `dbPromptForPassword`.
Specify the Backup Host and Directory

Vbr lists each node name for your cluster. Enter the backup host name and directory for each cluster node at the prompts:

Node v_vmart_node0001
Backup host name (no default):
Backup directory (no default):

No defaults exist for either the host name or the backup directory.

In the configuration file, all cluster nodes appear listed by name under the [Mapping] section. Each cluster node contains a line that includes the cluster node, backup host, and backup directory location:

[Mapping]
v_vmart_node0001 = 194.66.82.11:/home/dbadmin/backups

Save the Configuration File

To save the configuration file with its default name, enter a configuration file name of your choice, or press (Enter):

Config file name (fullbak.ini):

The default name consists of:

- The backup name you supplied (fullbak in this example)
- An .ini file extension

The vbr utility confirms that it saved the file:

Saved vbr configuration to fullbak.ini.

By default, Vertica saves the file in the current working directory. You can include a fully qualified or relative path to save the configuration file to an alternate location.

Note: The backup configuration file is typically stored on the cluster you are backing up. Therefore, Vertica recommends that you also save a copy of the configuration file on the backup host. If you do so, the configuration file remains available, even if the cluster node
is lost.

Continue to Advanced Settings

To continue with advanced configuration settings, enter y:

Change advanced settings? (n) [y/n]:

To continue without advanced settings and save the configuration file with the information you just entered, press Enter.

Changing the objectRestoreMode Parameter Value

After you create the configuration file, you can also specify an objectRestoreMode parameter. The objectRestoreMode parameter is associated only with restoring object-level backups.

Related Tasks

- Configuring Advanced VBR Options

Configuring Required VBR Parameters

To invoke the vbr utility to create a configuration file, enter this command:

> /opt/vertica/bin/vbr --setupconfig

The utility prompts you to answer the following questions. Each is followed by its default value, if any exists. Type Enter to accept a default value.

The second table column lists the name of the parameter associated with the question, as it appears in the configuration file.

<table>
<thead>
<tr>
<th>Setupconfig Prompt</th>
<th>Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snapshot name: (backup_snapshot)</td>
<td>snapshotName</td>
</tr>
<tr>
<td>Number of restore points? (1):</td>
<td>restorePointLimit</td>
</tr>
</tbody>
</table>
Specify objects (no default):
Object restore mode (coexist, createOrReplace or create)(createOrReplace):
Vertica user name (current user):
Save password to avoid runtime prompt (n) [y/n]
Database user password to save in vbr password config file (no default):
Backup host name (no default):
Backup directory (no default):
Config file name (backup_snapshot.ini):
Password file name (no default):
Change advanced settings? (n) [y/n]:

To change any advanced parameters, respond to the last question, Change advanced settings? by entering y.

After you successfully complete all of the required questions, vbr generates a configuration file with the information you supplied. Create separate configuration files for a full backup and each object-level backup. Use distinct backup names in each configuration file.

When the setup completes processing, enter a configuration file name. Use this file name when you run the --task backup or other commands. The utility uses the configuration file contents for both backup and restore tasks, and also for the --copycluster task.

If you do not successfully complete all of the required questions, vbr lists error messages and hints but does not create a configuration file. You can then run the --setupconfig command again to specify any missing or incorrect information.

Sample Session Configuring Required Parameters

The following sample test session shows the required configuration file parameters needed to create a configuration file. The utility detects the Vertica node names in the cluster, so you do not have to supply them (v_example_node0001, v_example_node0002, and v_exampledb_node0003):

```
> /opt/vertica/bin/vbr --setupconfig
Snapshot name (snapshotName): ExampleBackup
Backup Vertica configurations? (n) [y/n] y
Number of restore points? (1): 5
Specify objects (no default): dim, dim2
Object restore mode (coexist, createOrReplace or create) (createOrReplace): coexist
```
See Also

- VBR Configuration File Reference

Related Tasks

- Creating vbr Configuration Files
- Copying the Database to Another Cluster

Configuring Advanced VBR Options

When you use vbr to create your configuration file, you can configure advanced parameters after configuring the basic ones that are required. The first time you invoke this utility, enter the command:

```
> /opt/vertica/bin/vbr --setupconfig
```

After you complete the required parameter options, continue to advanced settings. Respond to the last question by entering y in response:

```
Change advanced settings? (n)[y/n]: y
```

The completed configuration file includes the following list of the advanced parameters, their description, and their default values:
<table>
<thead>
<tr>
<th><strong>Configuration Parameter</strong></th>
<th><strong>Description and Default Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>checksum</td>
<td>When set to y, this parameter uses checksum for data integrity instead of file date and size. The default value is n.</td>
</tr>
<tr>
<td>concurrency_backup</td>
<td>The maximum number of backup TCP rsync connections per node. The default value is 1.</td>
</tr>
<tr>
<td>concurrency_restore</td>
<td>The maximum number of restore TCP rsync connections per node. The default value is 1.</td>
</tr>
<tr>
<td>encrypt</td>
<td>When set to y, encrypts data during transmission. The default value is n.</td>
</tr>
<tr>
<td>port_rsync</td>
<td>The Port number for the rsync daemon. The default value is 50000.</td>
</tr>
<tr>
<td>retryCount</td>
<td>The number of times to retry if a connection attempt fails. The default value is 2.</td>
</tr>
<tr>
<td>retryDelay</td>
<td>The number of seconds to wait between connection retry attempts. The default value is 1.</td>
</tr>
<tr>
<td>serviceAccessUser</td>
<td>The user name used for simple authentication of rsync connections. Vertica stores the password for this user name in a separate configuration file.</td>
</tr>
<tr>
<td>tempDir</td>
<td>The temporary directory (/tmp/vbr). For additional information about the tempDir configuration parameter, see [Misc] Miscellaneous Settings.</td>
</tr>
<tr>
<td>total_bwlimit_backup</td>
<td>The total bandwidth limit in kilobytes per second per node for backup connections. Vertica distributes this bandwidth among the number of connections set in concurrency_backup. The total network load allowed by this value is the number of nodes multiplied by the value of this parameter. For example, a three node cluster and a total_bwlimit_backup value of 100 would allow 300Kbytes/sec of network traffic. The default value of 0 allows unlimited bandwidth.</td>
</tr>
<tr>
<td>total_bwlimit_restore</td>
<td>The total bandwidth limit in kilobytes per second per node for restore connections. Vertica distributes this bandwidth among the number of connections set in concurrency_restore. The total network load allowed by this value is the number of nodes multiplied by the value of this parameter. The default value of 0 allows unlimited bandwidth.</td>
</tr>
</tbody>
</table>
### Configuration Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description and Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>by the value of this parameter. For example, a three node cluster and a <code>total_bwlimit_restore</code> value of 100 would allow 300Kbytes/sec of network traffic. The default value of 0 allows unlimited bandwidth.</td>
</tr>
</tbody>
</table>

### Example of Configuring Advanced Parameters

This example shows how you might set some of the advanced configuration file parameters to configure your backup and restore settings:

```
Change advanced settings? (n)[y/n]: y
Temp directory (/tmp/vbr):
Number of times to retry backup? (2): 5
Seconds between retry attempts? (1): 3
Encrypt data during transmission? (n) [y/n] n
Use checksum for data integrity (not file date and size)? (n)[y/n]: n
Port number for Rsync daemon (50000):
User name to access rsync daemon (no default): rsyncaccount
Password of the user who accesses rsync daemon:
Transfer bandwidth limit in KBPS or 0 for unlimited (0): 0
```

Saved vbr configuration to exampleBackup.ini.

### See Also

- VBR Configuration File Reference

### Related Tasks

- Configuring Backup Hosts

### Configuring the Hard-Link Local Parameter

Creating hard link local backups requires that you manually add the `hardLinkLocal=True` parameter to the `[Transmission]` section of the `vbr` configuration file.
Any configuration file you generate with the `vbr --setupconfig` always includes a `[Transmission]` section. If you have an existing `vbr` configuration file (`backup_name.ini`), add the `hardLinkLocal` parameter to the `[Transmission]` section.

### Generating a Configuration File

To generate a configuration file *without* advanced options:

1. Add the parameter as the sole entry in the `[Transmission]` section:

   ```
   [Transmission]
   hardLinkLocal = True
   ```

2. Save the configuration file.

You can also generate a configuration file *with* advanced options.

1. Add the `hardLinkLocal` parameter as the last entry in the `[Transmission]` section:

   ```
   [Transmission]
   encrypt = False
   checksum = False
   port_rsync = 50000
   total_bwlimit_backup = 0
   total_bwlimit_restore = 0
   hardLinkLocal = True
   ```

2. Save the configuration file.

### Restrictions for the Backup Encryption Option

You cannot use `encrypt` parameter advanced option when creating a hard link local backup. If you add `hardLinkLocal=true` to a configuration file that includes `encrypt=true`, `vbr` issues a warning and then ignores the encryption parameter.

### Configuring a Local Backup File Without Hard Links

To create a local backup without hard file links, omit the `hardLinkLocal=True` parameter from the configuration file. Specify the `backupDir` parameter as a location on the same file system as the database catalog and data files. Then, the `vbr` utility creates a backup by copying the files, even when they are located on the same file system.
Example Password Configuration File

The following example password configuration file shows the options for configuring VBR passwords. If you have chosen not to store passwords, this file remains empty. Only the dbadmin or members of that user's permission group can view the contents of the file.

```
[Passwords]
dbPassword = DBsecurity
serviceAccessPass = rsyncpwd
; Specifies password for remote database. Used only for restoring to alternate cluster.
dest_dbPassword = DestinationPwd
```

Sample VBR .ini Files

To assist users with the task of configuring the vbr utility, Vertica includes sample configuration files that you can copy, edit, and deploy for your various vbr tasks. Vertica automatically installs these files at /opt/vertica/share/vbr/example_configs.

Vertica includes the following files:

- **Database Copy to an Alternate Cluster** - copycluster.ini
- **External Central Backup/Restore** - backup_restore_full_external.ini
- **Full Hardlink Backup/Restore** - backup_restore_full_hardlink.ini
- **Full Local Backup/Restore** - backup_restore_full_local.ini
- **Object-Level Local Backup/Restore** - backup_restore_object_local.ini
- **Object Replication to an Alternate Database** - replicate.ini
- **Full Backup and Restore to an Alternate Cluster** - restore_to_other_cluster.ini
- **Backup/Restore To Amazon S3** - backup_restore_s3.htm
Database Copy to an Alternate Cluster

; This sample vbr configuration file is configured for the copycluster vbr task.
; Copycluster supports full database copies only, not specific objects.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ---------------------------------------------------------------;
;;; BASIC PARAMETERS;;;
; ---------------------------------------------------------------;

[Mapping]
; For each node of the source database, there must be a [Mapping] entry specifying the corresponding
; hostname of the destination database node.
; !!Mandatory!! node_name = new_host/ip (no defaults)
v_exampledb_node0001 = new_host1.example
v_exampledb_node0002 = new_host2.example
v_exampledb_node0003 = new_host3.example
v_exampledb_node0004 = new_host4.example
; v_exampledb_node0001 = 10.0.90.17
; v_exampledb_node0002 = 10.0.90.18
; v_exampledb_node0003 = 10.0.90.19
; v_exampledb_node0004 = 10.0.90.20

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
; parameter to specify which database to copy.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
; section.
; dbPromptForPassword = True

; ---------------------------------------------------------------;
;;; ADVANCED PARAMETERS;;;
; ---------------------------------------------------------------;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
; lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between retry attempts, if a failure occurs.
; retryDelay = 1

; Full path to the password configuration file containing database password credentials
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; The maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all copycluster connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of copycluster TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username

External Central Backup/Restore

; This sample vbr configuration file shows full or object backup and restore to a separate remote
backup-host for each respective database host.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ; ; BASIC PARAMETERS ;
; ; ------------------------------ ;

[Mapping]
; !!Mandatory!! This section defines what host and directory will store the backup for each node.
; node_name = backup_host:backup_dir
; In this "parallel backup" configuration, each node backs up to a distinct external host.
; To backup all database nodes to a single external host, use that single hostname/IP address in each
entry below.
v_exampeldb_node0001 = 10.20.100.156:/home/dbadmin/backups
v_exampeldb_node0002 = 10.20.100.157:/home/dbadmin/backups
v_exampledb_node0003 = 10.20.100.158:/home/dbadmin/backups
v_exampledb_node0004 = 10.20.100.159:/home/dbadmin/backups
#v_exampledb_node0001 = example1.net1:/home/dbadmin/backups
#v_exampledb_node0002 = example1.net2:/home/dbadmin/backups
#v_exampledb_node0003 = example1.net3:/home/dbadmin/backups
#v_exampledb_node0004 = example1.net4:/home/dbadmin/backups

[Misc]
// Recommended!! Snapshot name. Object and full backups should always have different snapshot names.
// Backups with the same snapshotName form a time sequence limited by restorePointLimit.
// Valid characters: a-z A-Z 0-9 _
snapshotName = backup_snapshot

[Database]
// Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to backup/restore.
dbName = current_database

// If this parameter is True, vbr prompts the user for the database password every time.
// If False, specify the location of password config file in 'passwordFile' parameter in [Misc] section.
dbNamePromptForPassword = True

//; ;; ADVANCED PARAMETERS ;;;
//; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

[Misc]
// The temp directory location on all database hosts.
// The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl lockf locking.
tempDir = /tmp/vbr

// How many times to retry operations if some error occurs.
retryCount = 2

// Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
retryDelay = 1

// Specifies the number of historical backups to retain in addition to the most recent backup.
// 1 current + n historical backups
restorePointLimit = 1

// Full path to the password configuration file
// Store this file in directory readable only by the dbadmin
// (no default)
passwordFile = /path/to/vbr/pw.txt

// When enabled, Vertica confirms that the specified backup locations contain
// sufficient free space and inodes to allow a successful backup. If a backup
// location has insufficient resources, Vertica displays an error message and
// cancels the backup. If Vertica cannot determine the amount of available space
// or number of inodes in the backupDir, it displays a warning and continues
// with the backup.
// enableFreeSpaceCheck = True

// When performing a backup, replication, or copycluster, specifies the maximum
// acceptable difference, in seconds, between the current epoch and the backup epoch.
// If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all backup connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bandlimit_backup = 0

; The maximum number of backup TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bandlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username

Full Hardlink Backup/Restore

; This sample vbr configuration file shows backup and restore using hard-links to data files on each
database host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; ------------------------------------------
; ;; BASIC PARAMETERS ;;;
; ------------------------------------------

[Mapping]
; For each database node there must be one [Mapping] entry to indicate the directory to store the
backup.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default).
; node_name = backup_host:backup_dir
; Must use [] for hardlink backups
v_examedb_node0001 = []:/home/dbadmin/backups
v_examedb_node0002 = []:/home/dbadmin/backups
v_examedb_node0003 = []:/home/dbadmin/backups
v_examedb_node0004 = []:/home/dbadmin/backups
[Misc]
;!!Recommended!! Snapshot name. Object and full backups should always have different snapshot names.
; Backups with the same snapshotName form a time sequence limited by restorePointLimit.
; Valid characters: a-z A-Z 0-9 -
; snapshotName = backup_snapshot

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

[Transmission]
;!!Mandatory!! Identifies the backup as a hardlink style backup.
hardLinkLocal = True

; ;;; ADVANCED PARAMETERS ;;;
; -------------------------------------

[Database]
;!!Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to backup/restore.
; dbName = current_database

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl lockf locking.
; tempDir = /tmp/vbr

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile =

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Specifies the number of backup attempts after an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts if a failure occurs.
; retryDelay = 1

; When enabled, Vertica confirms that the specified backup locations contain sufficient free space and inodes to allow a successful backup. If a backup location has insufficient resources, Vertica displays an error message and cancels the backup. If Vertica cannot determine the amount of available space or number of inodes in the backupDir, it displays a warning and continues with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600
Full Local Backup/Restore

; This is a sample vbr configuration file for backup and restore using a file system on each database host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; -----------------------------------------------
;;; BASIC PARAMETERS ;;;
; -----------------------------------------------

[Mapping]
; !!Mandatory!! For each database node there must be one [Mapping] entry to indicate the directory to store the backup.
; node_name = backup_host:backup_dir
; use [] for localhost
v_exampledb_node0001 = []:/home/dbadmin/backups
v_exampledb_node0002 = []:/home/dbadmin/backups
v_exampledb_node0003 = []:/home/dbadmin/backups
v_exampledb_node0004 = []:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name
; Valid values: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

; -----------------------------------------------
;;; ADVANCED PARAMETERS ;;;
; -----------------------------------------------

[Misc]

; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if some error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When enabled, Vertica confirms that the specified backup locations contain
; sufficient free space and inodes to allow a successful backup. If a backup
; location has insufficient resources, Vertica displays an error message and
; cancels the backup. If Vertica cannot determine the amount of available space
; or number of inodes in the backupDir, it displays a warning and continues
; with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator
; dbUser = current_username

Full Backup and Restore to an Alternate Cluster

; This sample vbr configuration file shows full backup and restore to another cluster.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

; -------------------------------------------------------------------
; ; ; BASIC PARAMETERS ; ;
; -------------------------------------------------------------------

[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default)
; node_name = backup_host:backup_dir
v_exampledb_node0001 = new_host0001:/home/dbadmin/backups
v_exampledb_node0002 = new_host0002:/home/dbadmin/backups
v_exampledb_node0003 = new_host0003:/home/dbadmin/backups
v_exampledb_node0004 = new_host0004:/home/dbadmin/backups

[NodeMapping]
; !!Recommended!! This section is required if node names are different between source and destination databases.
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr prompts the user for database password every time.
; If False, specify location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

; --------------------------; ;
; ;; ADVANCED PARAMETERS ;;
; --------------------------;

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if some error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Full path to the password configuration file.
; Store this file in a directory only readable by the dbadmin.
; (no default)
; passwordFile =

; When enabled, Vertica confirms that the specified backup locations contain sufficient free space and inodes to allow a successful backup. If a backup location has insufficient resources, Vertica displays an error message and cancels the backup. If Vertica cannot determine the amount of available space or number of inodes in the backupDir, it displays a warning and continues with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Sets options for transmitting the data when using backup hosts.
; Changes the default port number for the rsync protocol.
; port_rsync = 50000
Object-Level Local Backup/Restore

; This sample vbr configuration file shows object-level backup and restore
; using a file system on each database host for that host's backup.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;); or pound signs (#).
; Option and values are separated by an equal sign.
; Only arguments marked as '!!Mandatory!!' must be specified explicitly.
; All commented parameters are set to their default value.

; ---------------------------------------------------------------
; ;; BASIC PARAMETERS ;;
; ---------------------------------------------------------------

[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Backup host name (no default) and Backup directory (no default)
; node_name = backup_host:backup_dir
; [] indicates backup to localhost
v_exampledb_node0001 = [ ]:/home/dbadmin/backups
v_exampledb_node0002 = [ ]:/home/dbadmin/backups
v_exampledb_node0003 = [ ]:/home/dbadmin/backups
v_exampledb_node0004 = [ ]:/home/dbadmin/backups

[Misc]
; !!Recommended!! Snapshot name. Object and full backups should always have different snapshot names.
; Backups with the same snapshotName form a time sequence limited by restorePointLimit.
; Valid values: a-z A-Z 0-9 - _
; snapshotName = backup_snapshot

; Specifies which tables and/or schemas to backup.
; !!Mandatory!! objects for an object backup
; Specifies which tables and/or schemas to copy. For tables, the containing schema defaults to public.
; (no default)
objects = mytable, myschema, myothertable

; Specifies how Vertica handles objects of the same name when restoring schema or table backups.
; objectRestoreMode = createOrReplace

[Database]
; !!!Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to backup/restore.
; dbName = current_database

; If this parameter is True, vbr will prompt user for database password every time.
; If changed to False, specify location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

; ;;; ADVANCED PARAMETERS ;;;
; --------------------------------------

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.
; retryDelay = 1

; Specifies the number of historical backups to retain in addition to the most recent backup.
; 1 current + n historical backups
; restorePointLimit = 1

; Full path to the password configuration file
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When enabled, Vertica confirms that the specified backup locations contain sufficient free space and inodes to allow a successful backup. If a backup location has insufficient resources, Vertica displays an error message and cancels the backup. If Vertica cannot determine the amount of available space or number of inodes in the backupDir, it displays a warning and continues with the backup.
; enableFreeSpaceCheck = True

; When performing a backup, replication, or copycluster, specifies the maximum acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Database]
; Vertica user name for vbr to connect to the database.
; This is rarely needed since dbUser is normally identical to the database administrator.
Object Replication to an Alternate Database

; This sample vbr configuration file shows the replicate vbr task.
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; An equal sign separates options and values.
; Specify arguments marked '!!Mandatory!!' explicitly.
; All commented parameters are set to their default value.

;-----------------------------------------------

;;; BASIC PARAMETERS ;;;
;-----------------------------------------------

[Mapping]
; There must be one [Mapping] section for all of the nodes in your database cluster.
; !!Mandatory!! Target host name (no default)
; node_name = new_host
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004
#v_exampledb_node0001 = 10.0.90.17
#v_exampledb_node0002 = 10.0.90.18
#v_exampledb_node0003 = 10.0.90.19
#v_exampledb_node0004 = 10.0.90.20

[Misc]
; Specifies which tables and/or schemas to copy. For tables, the containing schema defaults to public.
; !!Mandatory!! objects for replication. The replicate task does not support full database copy
; Use comma-separated list for multiple objects
; (no default)
objects = mytable, myschema, myothertable

; Specifies how Vertica handles objects of the same name when copying schema or tables.
; objectRestoreMode = createOrReplace

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this parameter to specify which database to replicate.
; dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc] section.
; dbPromptForPassword = True

; !!Mandatory!! These settings are all mandatory for replication. None of which have defaults.
dest_dbName = target_db
dest_dbUser = dbadmin
dest_dbPromptForPassword = True
; ----------------------------------------------------
;; ADVANCED PARAMETERS ;;
; ----------------------------------------------------

[Misc]
; The temp directory location on all database hosts.
; The directory must be readable and writeable by the dbadmin, and must implement POSIX style fcntl
lockf locking.
; tempDir = /tmp/vbr

; How many times to retry operations if an error occurs.
; retryCount = 2

; Specifies the number of seconds to wait between retry attempts, if a failure occurs.
; retryDelay = 1

; Full path to the password configuration file containing database password credentials
; Store this file in directory readable only by the dbadmin.
; (no default)
; passwordFile = /path/to/vbr/pw.txt

; When performing a backup, replication, or copycluster, specifies the maximum
; acceptable difference, in seconds, between the current epoch and the backup epoch.
; If the time between the current epoch and the backup epoch exceeds the value
; specified in this parameter, Vertica displays an error message.
; SnapshotEpochLagFailureThreshold = 3600

[Transmission]
; Changes the default port number for the rsync protocol.
; port_rsync = 50000

; Total bandwidth limit for all backup connections in KBPS, 0 for unlimited. Vertica distributes
; this bandwidth evenly among the number of connections set in concurrency_backup.
; total_bwlimit_backup = 0

; The maximum number of backup TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_backup = 1

; The total bandwidth limit for all restore connections in KBPS, 0 for unlimited
; total_bwlimit_restore = 0

; The maximum number of restore TCP rsync connection threads per node.
; Results vary depending on environment, but values between 2 and 16 are sometimes quite helpful.
; concurrency_restore = 1

[Database]
; Vertica user name for vbr to connect to the database.
; This is very rarely be needed since dbUser is normally identical to the database administrator.
; dbUser = current_username
Backup/Restore To Amazon S3

; This sample vbr configuration file shows backup to AWS s3 shared storage
; Section headings are enclosed by square brackets.
; Comments have leading semicolons (;) or pound signs (#).
; Option and values are separated by an equal sign.
; Only arguments marked as '!!Mandatory!!' must be specified explicitly.
; All commented parameters are set to their default value.

; ----------------------------------------------- ;
; ;; BASIC PARAMETERS ;;
; ----------------------------------------------- ;

[S3]
; This section replaces the [Mapping] section and is required to back up to s3

; !!Mandatory!! S3 bucket name (no default).
s3_backup_path = s3://backup_bucket/database_backup_path/

; !!Mandatory!! directory used to manage locking during a backup (no default). If the directory is
mounted on the initiator host, you should use "[]" instead of the local host name. The file system
must support POSIX flock.
s3_backup_file_system_path = []:/home/dbadmin/backup_locks_dir/
; s3_backup_file_system_path = otherhost.example:/home/dbadmin/backup_locks_dir/

; Specifies SSL encrypted transfer.
s3_encrypt_transport = True

; Specifies the number of threads for upload/download - backup
s3_concurrency_backup = 10

; Specifies the number of threads for upload/download - restore
s3_concurrency_restore = 10

[Misc]
; !!Recommended!! Snapshot name
; Valid values: a-z A-Z 0-9 - _
snapshotName = backup_snapshot

[Database]
; !!Recommended!! If you have more than one database defined on this Vertica cluster, use this
parameter to specify which database to backup/restore.
dbName = current_database

; If this parameter is True, vbr prompts the user for the database password every time.
; If False, specify the location of password config file in 'passwordFile' parameter in [Misc]
section.
dbPromptForPassword = True

; ----------------------------------------------- ;
; ;; ADVANCED PARAMETERS ;;
; ----------------------------------------------- ;

[Misc]
; The temp directory location on all database hosts.
VBR Configuration File Reference

The configuration file options are grouped into sections within the configuration file:

In This Section

[Misc] Miscellaneous Settings

This section collects basic settings, including the name and location of your backup. The section also indicates whether you are keeping more than a single backup file, as specified by the restorePointLimit parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Configuration Setting and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshotName</td>
<td>snapshotName</td>
<td>Forms the basis for a series of backups, up to the number of backups specified in restorePointLimit. Each iteration in this series consists of the snapshotName and a timestamp for a specific backup. Each series of backups should have a unique</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Configuration Setting and Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| tempDir                | /tmp/vbr  | Specifies an absolute path to a temporary storage area on the cluster nodes. The `tmp` path must be the same on all nodes in the cluster. The `vbr` utility uses this directory as a temporary location while it is copying files from the source cluster node to the destination backup location. Vertica also writes backup logs to this location. The `vbr` utility uses this directory as a temporary location for log files, lock files and other bookkeeping information during the execution of a task.  
Do not specify the same location as your database's data or catalog directory. Any unexpected files or directories in your data or catalog location can cause errors during database start or restore.  
The file system at this location must support `fcntl lockf (POSIX)` file locking. |
| restorePointLimit      | 1         | Specifies the number of historical backups to retain in addition to the most
### Parameter | Default | Configuration Setting and Description
---|---|---
| recent backup. For example, if you set restorePointLimit=3, Vertica saves three historical backups, in addition to the most recent backup, for a total of four backups. By default, Vertica maintains a current backup and one historical backup. Saving multiple backups lets you back up incrementally. Enter a positive integer. Saves multiple backups to the same location, which are shared through hard links. In such cases, listbackup displays the common backup prefix but indicates unique time and date suffixes: my_archive20111111_205841
| objects | None | Specifies whether vbr creates a full or object-level backup. If you do not specify any objects, vbr creates a full backup. Otherwise, specify the object names (schemas or tables) to include in a backup. To enter more than one object, enter multiple names in a comma-separated list. To identify objects using wildcards, use the includeObjects parameter instead. Enter table names in the form schema.objectname. For example, to make backups of the table customers from the schema finance, enter finance.customers. If a public table and a schema have the same name, vbr backs up only the schema. Use the schema.objectname convention to avoid confusion. Object names can include UTF-8 alphanumeric characters. Object names
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Configuration Setting and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cannot include escape characters, single quote (') or double quote (&quot;) characters. To use non-alphanumeric characters, use a backslash () followed by a hex value. For instance, if the table name is a my table (my followed by a space character, then table), enter the object name as follows: objects=my\20table This parameter also identifies objects that you want to replicate to an alternate cluster. This parameter conflicts with the includeObjects parameter. Use one or the other in your configuration file.</td>
</tr>
<tr>
<td>objectRestoreMode</td>
<td>createOrReplace</td>
<td>Specifies how Vertica handles objects of the same name when restoring schema or table backups. <strong>Valid Values:</strong> - createOrReplace - create - coexist For descriptions of these settings, refer to Creating vbr Configuration Files.</td>
</tr>
<tr>
<td>retryCount</td>
<td>2</td>
<td>Specifies the number of backup attempts to complete execution after an error occurs. If the backup fails after exceeding the number of retry attempts, the utility reports an error and stops processing.</td>
</tr>
<tr>
<td>retryDelay</td>
<td>1</td>
<td>Specifies the number of seconds to wait between backup retry attempts, if a failure occurs.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Configuration Setting and Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>passwordFile</td>
<td>None</td>
<td>Specifies the file name of the <strong>password configuration file</strong>.</td>
</tr>
<tr>
<td>enableFreeSpaceCheck</td>
<td>True</td>
<td>When enabled, Vertica confirms that the specified backup locations contain sufficient free space and inodes to allow a successful backup. If a backup location has insufficient resources, Vertica displays an error message and cancels the backup. If Vertica cannot determine the amount of available space or number of inodes in the backupDir, it displays a warning and continues with the backup. If you do not include this setting in your configuration file, Vertica performs the space check by default.</td>
</tr>
<tr>
<td>SnapshotEpochLagFailureThreshold</td>
<td>3600</td>
<td>When performing a backup, replication, or copycluster, specifies the maximum acceptable difference, in seconds, between the <strong>current epoch</strong> and the backup epoch. If the time between the current epoch and the backup epoch exceeds the value specified in this parameter, Vertica displays an error message. A value of 0 disables this check. If you do not include this setting in your configuration file, Vertica performs this check using the default value. Backups do not contain information committed after the backup epoch.</td>
</tr>
<tr>
<td>includeObjects</td>
<td>None</td>
<td>Identifies database objects and wildcard patterns to include with a backup task. Use a comma to delimit multiple objects and wildcard patterns. For more information, refer to <a href="#">Using Wildcards with Backup, Restore, and Replicate</a>. This parameter conflicts with the</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Configuration Setting and Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>objects</td>
<td></td>
<td>objects parameter. Use one or the other in your configuration file.</td>
</tr>
<tr>
<td>excludeObjects</td>
<td>None</td>
<td>Identifies database objects and wildcard patterns to exclude from a backup task. Use a comma to delimit multiple objects and wildcard patterns. For more information, refer to Using Wildcards with Backup, Restore, and Replicate. Use of this parameter requires use of includeObjects</td>
</tr>
</tbody>
</table>

### [Database] Database Access Settings

Sets options for accessing the database.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbName</td>
<td>N/A</td>
<td>Specifies the name of the database to back up. If you do not supply a database name, the vbr utility selects the current database to back up. OpenText recommends that you provide a database name.</td>
</tr>
<tr>
<td>dbUser</td>
<td>Current user name</td>
<td>Identifies the Vertica user used for database operations performed by vbr. In the case of the replicate task, this user is the source database user. The vbr utility obtains this information automatically as the current user of the person who invoked the --setupconfig command. You must be logged on as the database administrator to back up the database. The password, if you choose to save it, is stored in the Password Configuration File.</td>
</tr>
<tr>
<td>dbPromptForPassword</td>
<td>True</td>
<td>Controls whether the utility prompts for a password. If you set this parameter to False (indicating no prompt at run time), then you must</td>
</tr>
</tbody>
</table>
Vertica uses destination database parameters only to replicate objects to alternate clusters. By default, these parameters do not exist in your vbr configuration files. You must manually edit your database configuration file to add them.

### [Transmission] Data Transmission

Sets options for transmitting the data when using backup hosts.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt</td>
<td>False</td>
<td>Controls whether the transmitted data is encrypted while it is being copied to the target backup location. Choose this option if you are performing a backup over an untrusted network (for example, backing up to a remote host across the Internet).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Encrypting data transmission causes significant processing overhead and slows transfer. One of the processor cores of each database node is consumed during the encryption process. Use this option only if you are concerned about the security of the network used when transmitting backup data.</td>
</tr>
<tr>
<td>checksum</td>
<td>False</td>
<td>Controls whether the vbr utility has rsync use the md5 checksum to determine whether files are identical before and after network transmission. By default, rsync does not perform checksum. Instead, it performs minimal file checking, confirming that the file size and time of last modification are identical before and after transmission. <strong>Note:</strong> Calculating checksum values increases processor usage during the backup process.</td>
</tr>
<tr>
<td>port_rsync</td>
<td>50000</td>
<td>Changes the default port number for the rsync protocol. Change this value if the default rsync port is in use on your cluster, or you need rsync to use another port to avoid a firewall restriction.</td>
</tr>
<tr>
<td>total_bwlimit_backup</td>
<td>0</td>
<td>The total bandwidth limit in KBps for backup connections. Vertica distributes this bandwidth evenly among the number of connections set in concurrency_backup. The default value of 0 allows unlimited bandwidth.</td>
</tr>
<tr>
<td>concurrency_backup</td>
<td>1</td>
<td>The maximum number of backup TCP rsync connection threads per node. To improve local and remote backup, replication, and copy cluster performance, you can increase the number of threads available to perform backups. Increasing the number of threads allocates more CPU resources to the backup task and can, for remote backups, increase the amount of bandwidth used. The optimal value for this setting depends greatly on your specific configuration and requirements. Values higher than 16 produce no additional benefit.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>total_bwlimit_restore</td>
<td>0</td>
<td>The total bandwidth limit in KBps for restore connections. Vertica distributes this bandwidth evenly among the number of connections set in concurrency_restore. The default value of 0 allows unlimited bandwidth.</td>
</tr>
<tr>
<td>concurrency_restore</td>
<td>1</td>
<td>The maximum number of restore TCP rsync connections per node. The maximum number of restore TCP rsync connection threads per node. To improve local and remote restore, replication, and copy cluster performance, you can increase the number of threads available to perform restores. Increasing the number of threads allocates more CPU resources to the restore task and can, for restores of remote backups, increase the amount of bandwidth used. The optimal value for this setting depends greatly on your specific configuration and requirements. Values higher than 16 produce no additional benefit.</td>
</tr>
<tr>
<td>serviceAccessUser</td>
<td>None</td>
<td>The user name used for simple authentication of rsync connections. This user is neither a Linux nor Vertica user name. It is simply an arbitrary identifier used by the rsync protocol. If you do not provide a user name, Vertica leaves rsync running without authentication, creating a potential security risk. If you choose to save the password, Vertica stores it in the Password Configuration File.</td>
</tr>
<tr>
<td>hardLinkLocal</td>
<td>False</td>
<td>Creates a full- or object-level backup using hard file links on the local file system, rather than copying database files to a remote backup host. Add this configuration parameter manually to the Transaction section of the configuration file, as described in Configuring the Hard-Link Local Parameter.</td>
</tr>
<tr>
<td>port_ssh_backup</td>
<td>22</td>
<td>Overrides the default SSH port setting (22) for the backup hosts. Enter the required SSH port for your site.</td>
</tr>
</tbody>
</table>
### Parameter Default Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Changing the default SSH port is supported only when using the backup and restore tasks. Using a non-default SSH port with the copycluster task is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE:</strong> This parameter is not included in the configuration file automatically. For more information, see <a href="#">Configuring Backup Hosts</a>.</td>
</tr>
</tbody>
</table>

### [Mapping]

You have one [Mapping] section for all of the nodes in your database cluster. The section must appear in your configuration file because it specifies all database nodes being included in the backup. It also includes the backup host and directory for each node. If you have objects being replicated to an alternate database, the [Mapping] section also maps the target database nodes to the source database backup locations.

- If you edit an existing configuration file to add a Mapping in the current style, you must combine information from all existing Mappings into the new section.

- Alternatively, you can use `vbr` with the `--setupconfig` option to generate a new configuration file, as described in [Creating vbr Configuration Files](#).

**Note:** The [S3] and [Mapping] configuration sections are mutually exclusive. If you include both, your backup fails with the error message "Config has conflicting sections (Mapping, S3), specify only one of them."

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backupHost</td>
<td>None</td>
<td>Indicates the target host name or IP address on which to store this node's backup. The backupHost name is different from <code>dbNode</code>. The copycluster command uses this value to identify the target database node host name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Performance Consideration:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Although supported, backups to an NFS host may have poor performance, particularly on networks shared with rsync operations.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| backupDir  | None    | Identifies the full path to the directory on the backup host or node where the backup will be stored. **Directory Requirements:**
|            |         | • Must already exist when you run the utility with the `-task backup` option    |
|            |         | • Must be writable by the user account used to run the backup utility.         |
|            |         | • Must be unique to the database you are backing up. Multiple databases cannot share the same backup directory. |
|            |         | • The file system at this location must support `fcntl lockf` file locking.    |
| dbNode     | None    | The name of the database node, as recognized by Vertica. This value is not the node's host name, but rather the name Vertica uses internally to identify the node, usually in the form of: |
|            |         | `v_node00xx`                                                                 |
|            |         | To find database node names in your cluster, query the `node_name` column of the NODES system table. |

**Map to the localhost**

Vertica vbr does not support the special localhost name as a backup host. To backup a database node to its own disk, use empty square brackets for the hostname in the `[Mapping]` section of the configuration file.

```
[Mapping]
NodeName = []:/backup/path
```

Your mapping section should resemble this example:

```
[Mapping]
v_node0001 = []:/scratch_drive/archive/backupdir
```
Map to the Same Database

The following example shows how you can specify a Mapping section that indicates a single node to be backed up (v_vmart_node0001). The node is assigned to the backup host (v_srv01), and the backup directory (/home/dbadmin/backups). Although you are backing up a single node cluster, and the backup host and the database node are the same system, you specify them differently.

Specify the backup host and directory, using a colon (:) as a separator:

```
[Mapping]
v_vmart_node0001 = srv01:/home/dbadmin/backups
```

Although the configuration file [Mapping] section no longer uses named parameters, you still use the elements of the simplified format continue to represent the following parameters:

```
dbNode = backupHost:backupDir
```

Map to an Alternate Database

Before you can replicate objects to an alternate database, you must also create a [NodeMapping] section in your vbr configuration file. The NodeMapping section points source nodes to their target database nodes.

Restore an alternate database, by adding mapping information in the following form:

```
[Mapping]
targetNode: sourceDBNode_backuhost:sourceDB_backuppath
```

Your mapping section should resemble this example:

```
[Mapping]
v_sec_node0001 = pri_bsrv01:/archive/backup
v_sec_node0002 = pri_bsrv02:/archive/backup
v_sec_node0003 = pri_bsrv03:/archive/backup
```
[NodeMapping]

Vertica uses the node mapping section exclusively to restore a full backup from one database to another, different, database. Be sure to update the [Mapping] section of your configuration file to point your target Vertica nodes to their source backup locations.

Important: Vertica does not automatically generate the [NodeMapping] section of the vbr configuration file. You must edit the file manually to add a node mapping section. Refer to Full Backup and Restore to an Alternate Cluster as an example.

Use the following form to specify node mapping:

source_vertica_node = target_vertica_host

For example, you can use the following mapping to restore content from one 4-node database to an alternate 4-node database.

```
[NodeMapping]
v_exampledb_node0001 = new_host0001
v_exampledb_node0002 = new_host0002
v_exampledb_node0003 = new_host0003
v_exampledb_node0004 = new_host0004
```

[S3]

Sets options for storing backup data on Amazon S3.

Note: The [S3] and [Mapping] configuration sections are mutually exclusive. If you include both, your backup fails with the error message "Config has conflicting sections (Mapping, S3), specify only one of them."

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3_backup_file_system_path</td>
<td></td>
<td>Specifies the host and path that you are using to handle file locking during the backup process. Vertica must be able to create a passwordless ssh connection to the location that you specify here. To use a local NFS file system, specify a value of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s3_backup_file_system_path = []:path</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To use an EC2 instance, specify a value of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s3_backup_file_system_path = [host_name]:path</td>
</tr>
<tr>
<td>s3_backup_path</td>
<td></td>
<td>Specifies the S3 bucket name and backup path for the backup to S3. When you backup to S3, all nodes back up to same S3 bucket. You must create the backup location on S3 before performing a backup. This value takes the following form:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s3_backup_path = s3://backup_bucket/database_backup_path/</td>
</tr>
<tr>
<td>s3_encrypt_transport</td>
<td>True</td>
<td>When True, uses SSL encryption to encrypt data moving between your Vertica cluster and your S3 instance. If you are backing up or restoring from an Amazon EC2 cluster, you must set this parameter to True.</td>
</tr>
<tr>
<td>s3_concurrency_backup</td>
<td>10</td>
<td>The maximum number of concurrent backup threads for backup to S3.</td>
</tr>
<tr>
<td>s3_concurrency_restore</td>
<td>10</td>
<td>The maximum number of concurrent restore threads for restoring from S3.</td>
</tr>
<tr>
<td>s3_encrypt_at_rest</td>
<td>None</td>
<td>To enable at rest encryption of your backups to S3, specify a value of sse. For more information on encrypting your S3 backups, refer to Encrypting Backups to S3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value takes the following form:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s3_encrypt_at_rest = sse</td>
</tr>
<tr>
<td>s3_sse_kms_key_id</td>
<td>None</td>
<td>If you are using Amazon’s Key Management Security, use this parameter to provide your key ID. If you enable encryption and do not include this parameter, vbr uses SS3-S3 encryption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value takes the following form:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s3_sse_kms_key_id = &lt;key_id&gt;</td>
</tr>
</tbody>
</table>
Backup and Restore Utility Reference

This section provides reference information about both the vbr utility commands, and its associated configuration file parameters.

VBR Utility Reference

Allows you to back up and restore either the full database, or one or more schema and table objects of interest. You can also copy a cluster and list backups you created previously.

The utility is located in the Vertica binary directory (/opt/vertica/bin/vbr on most installations).

Syntax

```
/opt/vertica/bin/vbr { command }
... [ --archive timestamp ]
... [ --config-file file ]
... [ --debug level ]
... [ --nodes node1 [, noder, ]... ]
... [ --showconfig ]
```

Where `command` is one of the following:

<table>
<thead>
<tr>
<th>Full Command</th>
<th>Short Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>-h</td>
<td>Shows a brief usage guide for the command.</td>
</tr>
<tr>
<td>--showconfig</td>
<td></td>
<td>Displays the current configuration settings.</td>
</tr>
<tr>
<td>--setupconfig</td>
<td></td>
<td>Asks a series of questions and generates a configuration file.</td>
</tr>
<tr>
<td>--task { backup</td>
<td>-t</td>
<td>Performs the specified task:</td>
</tr>
<tr>
<td>7.2_upgrade</td>
<td></td>
<td>- 7.2_upgrade — Upgrades an existing 7.1 or earlier backup to the manifest-based backup introduced in version 7.2.</td>
</tr>
<tr>
<td>collect-garbage</td>
<td></td>
<td>- backup — Creates a full database or object-level backup, depending on configuration file</td>
</tr>
<tr>
<td>copycluster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>full-check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>init</td>
<td></td>
<td></td>
</tr>
<tr>
<td>listbackup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quick-check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quick-repair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
</tr>
<tr>
<td>replicate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full Command</td>
<td>Short Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>`</td>
<td>restore }</td>
<td>specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>collect-garbage</code> — Rebuilds the backup manifest and deletes any unreferenced objects in the backup location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>copycluster</code> — Copies the database to another Vertica cluster.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>full-check</code> — Verifies all objects listed in the backup manifest against file system metadata, outputting missing and unreferenced objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>init</code> — Creates a new backup directory, or prepares an existing one, for use and creates necessary backup manifests. You must perform this task before the first time you create a backup in a directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>listbackup</code> — Displays the existing backups associated with the configuration file you supply. View this display to get the name of a backup that you want to restore.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>quick-check</code> Confirms that all backed-up objects appear in the backup manifest. Outputs any discrepancies between objects in the backup location and objects listed in the backup manifest.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>quick-repair</code> — Builds a replacement backup manifest, based on storage locations and objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>remove</code> — Removes the specified backup or restore point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>replicate</code>— Copies objects from one cluster to an alternate cluster.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>restore</code> — Restores a full or object-level database backup. Requires the configuration file that created the backup.</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **--archive timestamp**                       | Used with the `--task restore` and `--task remove` commands. Specifies the timestamp of the backup to restore or remove:  
  > vbr --task restore --config-file myconfig.ini --archive=20160115_182640                                                                 |
| **-c file**                                   | Indicates the configuration file to use as an absolute or relative path to the location from which you start the backup utility. If no file exists, an error occurs and the utility cannot continue. |
| **--nodes node1[,]...**                       | Specifies any nodes, in a comma-separated list, on which to perform a vbr task. The nodes listed match the names in the Mapping section of the configuration file.  
  **Caution:** Do not try to restore the entire database cluster from a partial database backup created from a subset of the nodes. Data loss could result. |
<p>| <strong>--debug level</strong>                             | Specifies the level of debugging messages (from 0 to 3) that the vbr utility provides. Level 3 indicates verbose, while level 0, the default, indicates no messages.                                                      |
| <strong>--report-file path/filename</strong>               | Optional. Outputs a delimited JSON file that describes the results of the associated full backup integrity check or garbage collection task.                                                                 |
| <strong>--restore-objects objects</strong>                 | Specifies the individual objects to restore from a full or object-level backup. If you are using wildcards, use <code>--include-objects</code> and <code>--exclude-objects</code> instead.                                                   |
| <strong>--s3-force-init</strong>                           | Used with the <code>--task init</code> command. Forces the init task to succeed on S3 storage targets when there is an identity/lock file mismatch. For more information, refer to Creating Backups on Amazon S3. |
| <strong>--showconfig</strong>                              | The configuration values being used to perform the task, displayed in raw JSON format before vbr starts begins.                                                                                             |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--list-all</td>
<td>Used with the --task listbackup command. Displays a list of all backups stored on the hosts and paths listed in the specified configuration file.</td>
</tr>
<tr>
<td>--json</td>
<td>Used with the --task listbackup command. Displays a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.</td>
</tr>
<tr>
<td>--list-output-file \path/\filename</td>
<td>Used with the --task listbackup command. Outputs a file containing a JSON delimited list of all backups stored on the hosts and paths listed in the specified configuration file.</td>
</tr>
<tr>
<td>--dry-run</td>
<td>Used with the --task command. Performs a test run of the specified command without actually performing the task. You can use this command to evaluate the impact of a particular vbr command without actually performing that command. For example, you could see the size of a potential backup, or the objects contained in that backup. Any task performed with the dry-run parameter has no impact on your database.</td>
</tr>
<tr>
<td>--include-objects</td>
<td>Used with the --task command to identify specific database objects and wildcard patterns to include with a restore task. Use a comma to delimit multiple objects and wildcard patterns. For more information, refer to Using Wildcards with Backup, Restore, and Replicate. Do not use this parameter with the --restore-objects parameter.</td>
</tr>
<tr>
<td>--exclude-objects</td>
<td>Used with the --task command to identify specific database objects and wildcard patterns to exclude from a restore task. Use a comma to delimit multiple objects and wildcard patterns. For more information, refer to Using Wildcards with Backup, Restore, and Replicate.</td>
</tr>
</tbody>
</table>

**See Also**

- VBR Configuration File Reference
Related Tasks

- Configuring the Backup Script
- Restoring Full Database Backups
- Restoring Object-Level Backups

Password Configuration File Reference

The vbr utility automatically creates the password configuration file to store any saved passwords. Only the database administrator or members of that user’s permission group can view the contents of the file. If you alter the file permissions from their default values (xx0), Vertica backup or restore actions fail with an error message.

Vertica creates the password file, even if you do not save any passwords. In this case, the file exists but remains empty. Vertica stores this file in the same location that you choose to save your backup and restore configuration file.

[Passwords] Password Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| dbPassword    | None    | Identifies the database administrator's password. If you set dbPromptForPassword to False, enter a password. Doing so prevents you from being prompted at run time. You do not need to specify additional settings. **Empty strings:** You cannot enter an empty string for the dbPassword in the configuration file. Vertica does not recommend using an empty string as a superuser password. If you do so, you must:  
  1. Set the dbPromptForPassword parameter to True.  
  2. Leave the dbPassword option blank. |
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3. Enter the empty string at the prompt each time you run the backup utility.</td>
</tr>
<tr>
<td>serviceAccessPass</td>
<td>None</td>
<td>Identifies the password for the rsync user account.</td>
</tr>
<tr>
<td>dest_dbPassword</td>
<td>None</td>
<td>The password for the dest_dbuser Vertica account. Vertica uses this value for replication tasks only.</td>
</tr>
</tbody>
</table>

## When to Back up the Database

In addition to any guidelines established by your enterprise, Vertica recommends that you back up your database in the following circumstances:

**Before:**

- You upgrade Vertica to another release.
- You drop a partition.
- You add, remove, or replace nodes in your database cluster.

**After:**

- You load a large volume of data.
- You add, remove, or replace nodes in your database cluster.
- You recover a cluster from a crash.

**If:**

The epoch in the latest backup is earlier than the current ancient history mark.

**Note:** Always create a new full backup after adding, removing, or replacing nodes. When you restore a full database backup, you must restore to a cluster that is identical to the one on which you created the backup.

Ideally, schedule ongoing backups to back up your data. You can run the Vertica `vbr` from a `cron` job or other task scheduler.
Related Terms

- Ancient history mark (AHM).

**Configuring Backup Hosts**

The `vbr` utility lets you back up your database to one or more hosts (known as *backup hosts*), that can be outside of your database cluster.

You can use one or more backup hosts or a single S3 bucket to back up your database. Use the `vbr` configuration file to specify which backup host each node in your cluster should use.

Before you back up to hosts outside of the local cluster, configure the target backup locations to work with the `vbr` utility. The backup hosts you use must:

- Have sufficient backup disk space.
- Be accessible from your database cluster through SSH.
- Have passwordless SSH access for the Database Administrator account.
- Have either the Vertica rpm or Python 2.7 and rsync 3.0.5 or later installed.
- If you are using a stateful firewall, configure your `tcp_keepalive_time` and `tcp_keepalive_intvl` sysctl settings to use values less than your firewall timeout value.

**Configuring TCP Forwarding on Database Hosts**

VBR depends on TCP forwarding to forward connections from your database hosts to your Vertica nodes for backup and restore tasks. Database hosts require TCP forwarding on both the source and destination databases for copycluster and replication tasks. TCP forwarding is necessary for the following connections:

- SSH connections by the admin user running `vbr`
- SSH connections from the cluster node running `vbr`
- SSH connections to Vertica cluster nodes (both source and destination for copycluster and replication operations)
SSH connections to backup hosts do not require SSH forwarding.

If it is not already set by default, set AllowTcpForwarding = Yes in your sshd_config file. By default, this file is located at /etc/ssh/sshd_config. If TCP forwarding is not enabled, tasks requiring it fail with the following message:
"Errors connecting to remote hosts: Check SSH settings, and that the same Vertica version is installed on all nodes."

Creating Configuration Files for Backup Hosts

Create separate configuration files for full or object-level backups, using distinct names for each configuration file. Also, use the same node, backup host, and directory location pairs. Specify different backup directory locations for each database.

Note: For optimal network performance when creating a backup, Vertica recommends that you give each node in the cluster its own dedicated backup host.

Preparing Backup Host Directories

Before vbr can back up a database, you must prepare the target backup directory. Run vbr with a task type of init to create the necessary manifests for the backup process. You need to perform the init process only once. After that, Vertica maintains the manifests automatically.

Estimating Backup Host Disk Requirements

Wherever you plan to save data backups, consider the disk requirements for incremental backups at your site. Also, if you use more than one archive, multiple archives potentially require more disk space. Vertica recommends that each backup host have space for at least twice the database node footprint size. Follow this recommendation regardless of the specifics of your site's backup schedule and retention requirements.

To estimate the database size from the used_bytes column of the storage_containers system table:

```
VMart=> select sum(used_bytes) from storage_containers where node_name='v_mydb_node0001';
            sum
----------
     302135743
(1 row)
```
If your site uses multiple backup host locations, you can estimate the database size requirements per node. Use a query, such as the following, substituting a backup host name for `node_name`:

```
select node_name, sum(used_bytes) as size_in_bytes from v_monitor.storage_containers group by node_name;
```

### Making Backup Hosts Accessible

You must verify that any firewalls between the source database nodes and the target backup hosts allow connections for SSH and rsync on port 50000.

The backup hosts must be running identical versions of rsync and Python as those supplied in the Vertica installation package.

### Setting Up Passwordless SSH Access

To access a backup host, the database administrator must meet two requirements to run the `vbr` utility:

- Have an account on each backup host, with write permissions to the backup directory.
- Have passwordless SSH access from each database cluster host to the corresponding backup host.

How you fulfill these requirements depends on your platform and infrastructure.

SSH access among the backup hosts and access from the backup host to the database node is not necessary.

If your site does not use a centralized login system (such as LDAP), you can usually add a user with the `useradd` command or through a GUI administration tool. See the documentation for your Linux distribution for details.

If your platform supports it, you can enable passwordless SSH logins using the `ssh-copy-id` command to copy a database administrator's SSH identity file to the backup location from one of your database nodes. For example, to copy the SSH identity file from a node to a backup host named `backup01`:

```
> ssh-copy-id -i dbadmin@backup01
Password:
```
Try logging into the machine with "ssh 'dbadmin@backup01'". Then, check the contents of the ~/.ssh/authorized_keys file to verify that you have not added extra keys that you did not intend to include.

```
> ssh backup01
Last login: Mon May 23 11:44:23 2011 from host01
```

Repeat the steps to copy a database administrator's SSH identity to all backup hosts you use to back up your database.

After copying a database administrator's SSH identity, you should be able to log in to the backup host from any of the nodes in the cluster. You are not prompted for a password.

### Increasing the SSH Maximum Connection Settings for a Backup Host

If your configuration requires backing up multiple nodes to one backup host (n:1), increase the number of concurrent SSH connections to the SSH daemon (sshd). By default, the number of concurrent SSH connections on each host is 10, as set in the sshd_config file with the MaxStartups keyword. The MaxStartups value for each backup host should be greater than the total number of hosts being backed up to this backup host.

To increase the MaxStartups value:

1. Log on as root to access the config file.
2. Open the SSH configuration file (/etc/ssh/sshd_config) in a text editor.
3. If the MaxStartups line is commented out with #, delete that character. Replace the current value, which may be a single integer or three integers like 10:30:60 with the new value. For example, to back up a 50 node cluster to one machine, use a value of 60:

   MaxStartups 60

   For more information on configuring MaxStartups, refer to the man page for that parameter.
4. Save the file.
5. Reload the file using the following command:
   ```
sudo /etc/init.d/sshd reload
   If you are using Red Hat 7/CentOS 7, use the following command instead: sudo /bin/systemctl reload sshd.service
   ```
6. Exit from root.

See Also

- Backup Configuration Options
- Creating Backups on Amazon S3

Related Tasks

- Enable Secure Shell (SSH) Logins.
- Configuring Advanced VBR Options

**Using Wildcards with Backup, Restore, and Replicate**

Vertica supports the use of wildcard characters to include or exclude database objects from your backup, restore, and replication tasks. You can use wildcards in your vbr .ini file or as vbr command line parameters.

**Wildcards**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches 0 or more characters. Case insensitive.</td>
</tr>
<tr>
<td>\</td>
<td>Escapes the next character. To include a literal ? or * in your table or schema name, use the \ character immediately before the escaped character. To escape the \ character itself, use a double .</td>
</tr>
<tr>
<td>&quot;</td>
<td>Escapes the . character. To include a literal . in your table or schema name, wrap the character in double quotation marks.</td>
</tr>
</tbody>
</table>
Matching Schemas

Any pattern without a . character represents a schema. For example, the patterns

```
includeObjects = customer*,s?
```

would match any schema beginning with the word customer and any schema consisting of s and one letter.

When you backup or restore a schema without referencing any of the tables of that schema, vbr automatically includes all of the tables in that schema. If you include a schema by name alone, you cannot exclude individual tables from that schema. For example, the following is invalid syntax.

```
includeObjects = VMart
excludeObjects = VMart.?table?
```

You can exclude objects from an included schema by identifying the schema with the pattern `<schemaname>.*`. The following example shows a valid way to include a schema and exclude specific tables and patterns.

```
--include-objects 'VMart.*'
--exclude-objects 'VMart.sales,VMart.*account*'
```

Matching Tables

Any pattern including the . character represents a table. For example, in a configuration file, the following pattern:

```
includeObjects = sales.newclients,sales.??
```

would match any table named newclients, belonging to the sales schema and any table name with two characters, belonging to the sales schema.

You can also match all schemas and tables in a database or backup by using the pattern `*.`. For example, you could restore all of the tables and schemas in a backup using this command:

```
--include-Objects '*. *
```

Because a vbr parameter runs from the command line, you must enclose the wildcards in single quote marks to prevent Linux from misinterpreting them.
Testing Wildcard Patterns

You can test the results of any pattern by using the --dry-run parameter with a backup or restore command. Commands that include --dry-run do not affect your database. Instead, vbr displays the result of the command without executing it. For more information on --dry-run, refer to the VBR Utility Reference.

Using Wildcards with Backups

You can identify objects to include in your object backup tasks using the includeObjects and excludeObjects parameters in your configuration file. A typical configuration file might include this content:

```plaintext
[Misc]
snapshotName = dbobjects
restorePointLimit = 1
enableFreeSpaceCheck = True
includeObjects = VMart.*,online_sales.*
excludeObjects = *.temp*
```

In this example, the backup would include all tables from the VMart and online_sales schemas, while excluding any table containing the string 'temp' in its name belonging to any schema.

After it evaluates included objects, vbr evaluates excluded objects and removes excluded objects from the included set. If, for example, you included schema1.table1 and then excluded schema1.table1, that object would be excluded. If no other objects were included in the task, the task would fail. The same is true for wildcards. If an exclusion pattern removes all included objects, the task fails.

Using Wildcards with Restore

You can identify objects to include in your restore tasks using the --include-objects and --exclude-objects vbr parameters.

**Note:** Take extra care when using wildcard patterns to restore database objects. Depending on your object restore mode settings, restored objects can overwrite your existing objects. Test the impact of a wildcard restore with the --dry-run vbr parameter before performing the actual task.
Vbr evaluates excluded objects after it evaluates included objects and removes excluded objects from the included set. If, for example, you included schema1.table1 and then excluded schema1.table1, that object would be excluded. If no other objects were included in the task, the task would fail.

A typical restore command might include this content:

```bash
vbr -t restore -c verticaconfig --include-objects 'customers.*,sales??'
--exclude-objects 'customers.199?,customers.200?'
```

This example would include the schema customers, minus any tables with names matching 199 and 200 plus one character, as well as all any schema matching 'sales' plus two characters.

Another typical restore command might include this content.

```bash
vbr -t restore -c replicateconfig --include-objects '*.transactions,flights.*'
--exclude-objects 'flights.DTW*,flights.LAS*,flights.LAX*'
```

This example would include any table named transactions, regardless of schema, and any tables beginning with DTW, LAS, or LAX belonging to the schema flights. Although these three letter airport codes are capitalized in the example, vbr is case insensitive.

## Types of Backups

The vbr utility supports several kinds of backups:

- **Full backups**
- **Object-level backups**
- **Hard-link local backups**

You can refer to full or object-level backups by a user-defined descriptive name, such as FullDBSnap, Schema1Bak, or Table1Bak.

## Full Backups

A full backup is a complete copy of the database catalog, its schemas, tables, and other objects. This type of backup provides a consistent image of the database at the time the backup occurred. You can use a full backup for disaster recovery to restore a damaged or incomplete database. You can also restore individual objects from a full backup.
When a full backup exists, vbr creates incremental backups as successive backups following the full-backup. Incremental backups contain new or changed data since the last full, or incremental, backup occurred.

Archives contain a collection of same-name backups. Each archive can have a different retention policy. For example, suppose that TBak is the name of an object-level backup of table T, and you create a daily backup each week. These seven backups become part of the TBak archive. Keeping a backup archive lets you revert back to any one of the saved backups.

Full and object-level backups reside on backup hosts, the computer systems on which backups and archives are stored.

Vertica saves backups in a specific backup location, the directory on a backup host. This location can contain multiple backups, including associated archives. The backups are also compatible, allowing you to restore any objects from a full database backup.

Object-Level Backups

An object-level backup consists of one or more schemas or tables or a group of such objects. The conglomerate parts of the object-level backup do not contain the entire database. When an object-level backup exists, you can restore all of its contents or individual objects.

Note: Vertica does not support object level backups on Hadoop Distributed File System (HDFS) storage.

Object-level backups contain the following object types:

<table>
<thead>
<tr>
<th>Selective objects</th>
<th>Objects you choose to be part of an object-level backup. For example, if you specify tables T1 and T2 to include in an object-level backup, they are the selected objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependent objects</td>
<td>Objects that must be included as part of an object-level backup, due to dependencies. Suppose you want to create an object-level backup that includes a table with a foreign key. To do so, table constraints require that you include the primary key table, and vbr enforces this requirement. Projections anchored on a table in the selected objects are also dependent objects.</td>
</tr>
<tr>
<td>Principal objects</td>
<td>The objects on which both selected and dependent objects depend are called principal objects. For example, each table and projection has an owner, and each is a principal object.</td>
</tr>
</tbody>
</table>

Vertica 6.x and later provides support for object-level backups.
Hard Link Local Backups

A hard-link local backup can be a full- or object-level backup. It consists of a complete copy of the database catalog, and a set of hard file links to corresponding data files. You must save a hard-link local backup on the file system used by the catalog and database files.

Backup Contents

Backups contain all committed data for the backed up objects as of the start time of the backup. Backups do not contain uncommitted data or data committed during the backup. Backups do not delay mergeout or load activity.

Related Tasks

- Configuring Backup Hosts

Creating Backups on Amazon S3

Vertica supports the creation of backups on Amazon S3 Standard cloud storage. You can create these backups from your local cluster or from Amazon EC2 virtual servers.

Note: Vertica supports backup and restore from S3. Copycluster and replication to S3 targets is not supported.

Creating an S3 Configuration File

To backup to Amazon S3, you must add an [S3] section to your backup configuration file. For more information, refer to [S3]. Vertica also provides a sample S3 configuration file that you can copy and edit.
Configuring Amazon S3 Storage for Backup

Vertica supports using Amazon S3 cloud storage as a backup location. As with all Vertica backups, Vertica creates incremental backups, meaning each subsequent backup contains only changes that have occurred since you created the first backup. As with any storage location, you must initialize an S3 storage location with the vbr task init.

Because S3 storage does not support file locking, Vertica uses either your local file system or an Amazon EC2 file system to handle file locks during a backup. You identify this location using the s3_backup_file_system_path parameter in your vbr configuration file. During a backup, Vertica creates a locked identity file on your local or EC2 instance, and a duplicate file in your S3 backup location. As long at the files match, Vertica proceeds with the backup, releasing the lock once the backup is complete. As long as the files remain identical, you can use the S3 location for backup and restore tasks.

If the files in your locking location become out of sync with the files in your backup location, backup and restore tasks fail with an error message. You can resolve locking inconsistencies by rerunning the init task with the --s3-force-init parameter.

A typical S3 locking file reset command takes the following form:

```
/opt/vertica/bin/vbr --task init --s3-force-init -c filename.ini
```

Note: If a backup fails, confirm that your Vertica cluster has permission to access your S3 storage location.

Configuring EC2 Authentication for Amazon S3

If you are backing to S3 from an EC2-based cluster, you must provide authentication to your S3 host. Regardless of the authentication type you choose, your credentials do not leave your EC2 cluster. Vertica supports the following authentication types:

- AWS credential file
- IAM role
- Environmental variables

AWS credential file - You can manually create a configuration file on your EC2 initiator host at ~/.aws/credentials.
For more information on credential files, refer to Amazon's Web Services documentation.

IAM role - Create an AWS IAM role and grant that role permission to access your EC2 cluster and S3 resources. For more information, refer to Amazon's Web Services documentation.

Environmental variables - (Recommended) Amazon Web Services provides the following environmental variables:

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

Use these variables on your initiator to provide authentication to your S3 host. Once your session ends, AWS deletes these variables. For more information, refer to the AWS documentation.

### Encrypting Backups to S3

Backups made to Amazon S3 can be encrypted using native server-side S3 encryption capability. For more information on Amazon S3 encryption, refer to Amazon documentation.

*Note: Vertica supports server-side encryption only. Client-side encryption is not supported.*

### Supported Encryption Types

Vertica supports the following forms of S3 encryption:

**Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)**

- Encrypts backups with AES-256
- Amazon manages encryption keys

**Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)**

- Encrypts backups with AES-256
- Requires an encryption key from Amazon Key Management Service
- Your S3 bucket must be from the same region as your encryption key
- Allows auditing of user activity

### Configuring Amazon S3 for Encrypted Backups

When you enable encryption of your backups, Vertica encrypts backups as it creates them. If you enable encryption after creating an initial backup, only increments added after you enabled encryption are encrypted. To ensure that your backup is entirely encrypted, create new backups after enabling encryption.

To enable encryption, add the following settings to your configuration file:

- **s3_encrypt_transport** - Encrypts your backups during transmission. You must enable this parameter if you are using SSE-KMS encryption.
- **s3_encrypt_at_rest** - Enables encryption of your backups. If you enable encryption and do not provide a KMS key, Vertica uses SSE-S3 encryption.
- **s3_sse_kms_key_id** - If you are using KMS encryption, use this parameter to provide your key ID.

For more information on these settings, refer to [S3 configuration settings](#).

The following example shows a typical configuration for KMS encryption of backups.

```ini
[S3]
s3_encrypt_transport = True
s3_encrypt_at_rest = sse
s3_sse_kms_key_id = 6785f412-1234-4321-8888-6a774ba2aaaa
```

### Backing up and Restoring from Encrypted S3

You can create and restore encrypted backups from S3 just as you would any other backup.

### Creating Object-Level Backups

The database administrator can create object-level backups consisting of one or more schemas and tables. Object-level backups are especially useful for multi-tenanted database sites. For example, an international airport could use a multi-tenanted database to represent different
airlines in its schemas. Then, tables could maintain various types of information for the airline, including ARRIVALS, DEPARTURES, and PASSENGER information. With such an organization, creating object-level backups of the specific schemas would let you restore by airline tenant, or any other important data segment.

The following configuration file parameters are specific to object-level backups:

- `includeObjects`
- `excludeObjects`
- `objectRestoreMode` (used in restore operations only)

For more information about creating configuration files for full or object-level backups, see Configuring Required VBR Parameters.

**Note:** Apache Kafka uses internal configuration settings to maintain the integrity of your data. When backing up your Kafka data, Vertica recommends that you perform a full database backup, rather than an object-level backup.

### Preparing Your Backup Directory

Before you can create a backup, you must prepare your backup directory with the `vbr-init` task.

### Invoking vbr Backup

After creating the configuration file specifying which objects to backup, you can create an object-level backup. The following example shows how you can use the `objectbak.ini` command in a configuration file:

```
[dbadmin@v_vmart_node0001 ~]$ vbr --task backup --config-file objectbak.ini
Preparing...
Found Database port: 5433
Copying...
[==================================================] 100%
All child processes terminated successfully.
Committing changes on all backup sites...
backup done!
[dbadmin@v_vmart_node0001 ~]$
```
Backup Locations and Naming

You can use one top-level backup directory to store both full and object-level backups.

Note: Vertica does not recommend concurrent backups. If you must run multiple backups concurrently, use separate backup and temp directories for each. Having separate backup directories detracts from the advantage of sharing data among incremental backups.

To see existing full and object-level backups, see Viewing Backups in the See Also section below.

Best Practices for Object-Level Backups

To create one or more object-level backups, create a configuration file specifying the backup location, the object-level backup name, and a list of objects to include (one or more schemas and tables). When creating configuration backup files:

- Create one configuration file for each object-level backup
- Create a different configuration file to create a full database backup
- For best network performance, use one backup host per cluster node
- Use one directory on each backup-node to store successive backups
- For future reference, append the major Vertica version number to the configuration file name (mybackup7x)

Using the same backup host directory location for full and object-level backups results in the backups sharing disk space. Shared disk space makes backups compatible when performing a restore. Each cluster node must also use the same directory location on its designated backup host.

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema S1 and tables T1 and T2 in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).
Naming Conventions

Give each object-level backup configuration file a distinct and descriptive name. For instance, at an airport terminal, schema-based backup configuration files use a naming convention with an airline prefix, followed by further description, such as:

AIR1_daily_arrivals_backup
AIR2_hourly_arrivals_backup
AIR2_hourly_departures_backup
AIR3_daily_departures_backup

When database and object-based backups exist, you can recover the backup of your choice.

Caution: Do not change object names in an object-level configuration file if a backup already exists. Doing so overwrites the original configuration file, and you cannot restore it from the earlier backup. Instead, create a different configuration file.

Determining Backup Frequency

Vertica recommends, as a best practice, that you take frequent backups if database contents diverge in significant ways.

Always take backups after any event that significantly modifies the database, such as performing a rebalance. Mixing many backups with significant differences can weaken data K-safety. For example, taking backups both before and after a rebalance is not a recommended practice in cases where the backups are all part of one archive.

Understanding Object-Level Backup Contents

Object-level backups comprise only the elements necessary to restore the schema or table, including the selected, dependent, and principal objects. An object-level backup includes the following contents:

- **Storage:** Data files belonging to any specified objects
- **Metadata:** Including the cluster topology, timestamp, epoch, AHM, and so on
Catalog snippet: Persistant catalog objects serialized into the principal and dependent objects

Some of the elements that comprise AIR2, for instance, include its parent schema, tables, named sequences, primary key and foreign key constraints, and so on. To create such a backup, vbr script saves the objects directly associated with the table. It also saves any dependencies, such as foreign key (FK) tables, and creates an object map from which to restore the backup.

Note: Because the data in local temp tables persists only within a session, local temporary tables are excluded when you create an object-level schema. For global temporary tables, vbr stores the table's definition.

Making Changes After an Object-Level Backup

Be aware how changes made after an object-level backup affect subsequent backups. Suppose you create an object-level backup and later drop schemas and tables from the database. In this case, the objects you dropped are also be dropped from subsequent backups. If you do not save an archive of the object backup, such objects could be lost permanently.

Changing a table name after creating a table backup does not persist after restoring the backup. Suppose that, after creating a backup, you drop a user who owns any selected or dependent objects in that backup. In this case, restoring the backup re-creates the object and assigns ownership to the user performing the restore. If the owner of a restored object still exists, that user retains ownership of the restored object.

To restore a dropped table from a backup:

1. Rename the newly created table from t1 to t2.
2. Restore the backup containing t1.
3. Restore t1. Tables t1 and t2 now coexist.

For information on how Vertica handles object overwrites, refer to "Specifying Object Restore Mode" in Creating vbr Configuration Files.

K-safety may increase after an object backup. Restoration of a backup fails if both of the following conditions occur:

- An increase in K-safety occurs.
- Any table in the backup has insufficient projections.
Changing Principal and Dependent Objects

If you create a backup and then drop a principal object, restoring the backup restores that principal object. If the owner of the restored object has also been dropped, Vertica assigns the restored object to the current dbadmin.

You can specify how Vertica handles object overwrites in the `vbr` configuration file. For more information, refer to "Specifying Object Restore Mode" in Creating `vbr` Configuration Files.

Identity and auto-increment sequences are dependent objects because they cannot exist without their tables. An object-level backup includes such objects, along with the tables on which they depend.

Named sequences are not dependent objects because they exist autonomously. A named sequence remains after you drop the table in which the sequence is used. In this case, the named sequence is a principal object. Thus, you must back up the named sequence with the table. Then, you regenerate it, if it does not already exist when you restore the table. If the sequence does exist, `vbr` uses it, unmodified. Sequence values could repeat, if you restore the full database and then restore a table backup to a newer epoch.

Considering Constraint References

You must backup all database objects that are related through constraints. For example, suppose you have a schema with tables whose constraints reference only tables in the same schema can be backed up. However, a schema containing a table with an FK/PK constraint on a table in another schema cannot. To back up the second table, you must include the other schema in the list of selected objects.

Configuration Files for Object-Level Backups

The `vbr` utility automatically associates configurations with different backup names but uses the same backup location.

Always create a cluster-wide configuration file and one or more object-level configuration files pointing to the same backup location. Storage between backups is shared, preventing multiple copies of the same data. For object-level backups, using the same backup location causes `vbr` to encounter fewer OID conflict prevention techniques. Avoiding OID conflict prevention results in fewer problems when restoring the backup.
By using cluster and object configuration files with the same backup location, the utility includes additional provisions to ensure that the object-level backups can be used following a full cluster restore. One approach to restoring a full cluster is to use a full database backup to bootstrap the cluster. After the cluster is operational again, you can restore the most recent object-level backups for schemas and tables.

Attempting to restore a full database using an object-level configuration file fails, resulting in this error:

```
Note:
VMart=> /tmp/vbr --config-file=Table2.ini -t restore
Preparing...
Invalid metadata file. Cannot restore.
restore failed!
```

**Backup Epochs**

Each backup includes the epoch to which its contents can be restored. When `vbr` restores data, Vertica updates to the current epoch.

The `vbr` utility attempts to create an object-level backup five times before an error occurs and the backup fails.

**See Also**

- Creating `vbr` Configuration Files
- VBR Configuration File Reference
- Configuring Required VBR Parameters
- Viewing Backups
- Types of Backups
- Using Wildcards with Backup, Restore, and Replicate
Related Tasks

- Configuring Required VBR Parameters
- Restoring Object-Level Backups

Creating Hard-Link Local Backups

Before creating a full hard-link local database backup, verify the following:

- Your database is running. All nodes need not be up in a K-safe database for vbr to run. However, be aware that any nodes that are DOWN are not backed up.
- The user account that starts the utility (dbadmin or other) has write access to the target backup directories.

When you create a full or object-level hard link local backup, that backup contains the following contents.

<table>
<thead>
<tr>
<th>Backup</th>
<th>Catalog</th>
<th>Database files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full backup</td>
<td>Full copy</td>
<td>Hard-file links to all database files</td>
</tr>
<tr>
<td>Object-level</td>
<td>Full copy</td>
<td>Hard-file links for all objects listed in the configuration file, and any of their dependent objects</td>
</tr>
</tbody>
</table>

Run the vbr script from a terminal using the database administrator account from a node in your database cluster. You cannot run the utility as root.

To create a full or object-level backup, enter the following command:

```
> /opt/vertica/bin/vbr --task backup --config fullbak.ini
```

Note: While not required, Vertica recommends that you first create a full backup before creating any object-level backups.
Common Errors in Specifying the Hard-Link Local Backup Location

When you specify the hard-link backup location, be sure to avoid these common errors when adding the hardLinkLocal=True parameter to the configuration file:

<table>
<thead>
<tr>
<th>If ...</th>
<th>Then...</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>You specify a backup directory on a different node</td>
<td>vbr issues an error message and stops processing the backup.</td>
<td>Change the configuration file to include a backup directory on the same host and file system as the database and catalog files. Then, run the backup utility again.</td>
</tr>
<tr>
<td>You specify both of the following:</td>
<td>vbr issues a warning message</td>
<td>No action required. No action required.</td>
</tr>
<tr>
<td>• A backup destination directory on a different file system from the database and catalog files.</td>
<td>Performs backup by copying the files on the node from one file system to the other.</td>
<td></td>
</tr>
<tr>
<td>• That backup destination is on the same node.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creating Hard-Link Local Backups for Tape Storage

You can use hard-link local backups as a staging mechanism to backup to tape or other forms of storage media. You can also use the hard-link local backup to restore the hard file links to
the database files.

The following steps present a simplified approach to saving, and then restoring, hard-link local backups from tape storage:

1. Create a configuration file using a command such as:

   ```
   /opt/vertica/bin/vbr --setupconfig
   ```

2. Edit the configuration file (localbak.ini in this example) to include the hardLinkLocal=True parameter in the [Transmission] section.

3. Run the backup utility with the configuration file:

   ```
   /opt/vertica/bin/vbr --task backup --config-file localbak.ini
   ```

4. Copy the hard-link local backup directory with a separate process (not vbr) to tape or other external media.

5. If the database becomes corrupted, create the directory structure that existed when you created the hard-link local backup.

6. Transfer the backup files from tape to their original backup directory.

7. Using the configuration file you used to create the hard-link local backup (Step 3), restore the database using the following command:

   ```
   /opt/vertica/bin/vbr --task restore --config-file localbak.ini
   ```

   When you restore from a hard-link local backup (copied from tape), vbr creates hard links from the backup files to the database directory, if possible. This approach saves significant disk space and time.

**Related Terms**

- K-safe
- database administrator
Creating Full Backups

Before you create a database backup, verify the following:

- You have prepared your backup directory with the `vbr-init task`.

- Your database is running. It is unnecessary for all nodes to be up in a K-safe database. However, any nodes that are DOWN are not backed up.

- All of the backup hosts are up and available.

- The backup host (either on the database cluster or elsewhere) has sufficient disk space to store the backups.

- The user account of of the user who starts the utility has write access to the target directories on the host backup location. This user can be the dbadmin or another assigned role. However, you cannot run the utility as root.

- Each backup has a unique file name.

Run the `vbr` script from a terminal. Use the database administrator account from an initiator node in your database cluster.

Running vbr Without Optional Commands

You can run the `vbr` using only its required commands:

- `--task backup`

- `--config-file config_file`

If your configuration file does not contain the database administrator password, `vbr` prompts you to enter the password. However, it does not display what you type.

The utility requires no further interaction after you invoke it.

To run the `vbr` utility:

Use the `--task backup` and `--config-file filename` directives as shown in this example:

```
[release@qco5srv01:/scratch_b/qa/vertica/QA/VT_Scenario 0]$ vbr -t backup --config $FULLBAK_CONFIG
Starting backup of database VTDB.
```
Participating nodes: v_vmart_node0001, v_vmart_node0002, v_vmart_node0003, v_vmart_node0004.

Snapshots database.
Snapshot complete.
Approximate bytes to copy: 2315056043 of 2356089422 total.
[==================================================] 100%
Copying backup metadata.
Finalizing backup.
Backup complete!

By default, no output appears, other than the progress bar. To include additional progress information, use the --debug option, with a value between 1–3.

If the utility does not locate the configuration you specify, it searches for one at opt/vertica/config/vbr.ini. If no file exists, the backup fails with an error.

Best Practices for Creating Backups

When creating backup configuration files:

- Create separate configuration files to create full- and object-level backups.
- Use the same backup host directory location for both kinds of backups:
  - Because the backups share disk space, they are compatible when performing a restore.
  - Each cluster node must also use the same directory location on its designated backup host.
- For best network performance, use one backup host per cluster node.
- Use one directory on each backup node to store successive backups.
- For future reference, append the major Vertica version number to the configuration file name (mybackup7x).

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema S1 and tables T1 and T2 in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).

Incremental Backups

Each time you back up your database with the same configuration file, vbr creates an incremental backup. This incremental backup copies new storage containers, which can
include:

- Data that existed the last time you performed a database backup
- New and changed data since the last full backup

You can configure the `restorePointLimit` parameter to increase the number of stored backups.

**vbr Process for Deleting Backups**

Running the `vbr` utility with the `--task backup` command deletes the oldest backup. This command run whenever the total number that exist exceeds the `restorePointLimit` value in the configuration file. Suppose the `restorePointLimit` = 5, and five archives exist. In this case, running the `vbr --task backup` utility again deletes the backup with the oldest `date_timestamp`, when `vbr` completes the current backup command.

When you invoke `vbr` to create a backup:

1. The utility obtains the value of the `restorePointLimit` parameter value to determine how many backups should exist in the archive.
2. If creating the next backup exceeds the restore point limit, `vbr` deletes the oldest archived backup.
3. `vbr` continues processing and initially creates the backup on the database cluster.
4. When the new backup is complete, `vbr` copies it from the database cluster to the designated backup location.
5. After the new backup is successfully copied to the backup location, `vbr` removes the backup from the database cluster.

**See Also**

- Viewing Backups.
Interrupting the Backup Utility

To cancel a backup, use Ctrl+C or send a SIGINT to the Python process running the backup utility. The utility stops the backup procedure after it has completed copying the data. Canceling a `vbr` backup with Ctrl+C immediately closes the session. Uninterrupted incremental backups may run more quickly as a result.

The files generated by an interrupted backup process remain in the target backup location directory. The next backup process picks up where the interrupted process left off.

Backup operations are atomic, so that interrupting a backup operation does not affect the previous backup. Vertica replaces the previous backup only as the very last step of backing up your database.

The `restore` or `copy-cluster` operations overwrite the database catalog directory. Thus, interrupting either of these processes leaves the database unusable until you restart the process and allow it to finish.

Backup and Restore Resource Allocation

Vertica allows administrators to allocate bandwidth and TCP rsync connection resources to backup and restore operations. By default, Vertica allows a single connection and unlimited bandwidth for any backup or restore operation. By changing the default allocations in your `vbr` configuration file, you can:
- Increase the number of available connections. If you have many Vertica files, more connections can provide a significant performance boost as each connection increases the number of concurrent file transfers.

- Assign a limit to the total amount of bandwidth that a particular operation uses. If your files are fewer and larger, additional bandwidth can provide a performance boost.

For more information, refer to the `total_bwlimit_backup`, `total_bwlimit_restore`, `concurrency_backup`, and `concurrency_restore` settings in Configuring Advanced VBR Options.

### Increasing Queue Transport

Each connection creates an additional process that can transport an additional queue of files. When you start a backup or restore operation, Vertica creates a list of all the files requiring transport. As a connection completes a transfer, it takes the next file on the list until the list has been completed. If any file fails to transfer, Vertica assigns that file to another process.

The `retryCount` setting in the `vbr` configuration file determines the number of times `vbr` attempts to transfer a file. If the number of failures for a particular file exceeds the limit specified in the `retryCount` setting, the entire operation fails. `Vbr` then terminates processing.

### Bandwidth Limits

Vertica can limit its network bandwidth use through the `total_bwlimit_backup` and `total_bwlimit_restore` data transmission parameters. For more information, refer to [Transmission] Data Transmission.

### Viewing Backups

You can view backups in any of three ways:

- Use `vbr` to list the backups that reside on the local or remote backup host (requires a configuration file).
- View historical information about backups using the DATABASE_BACKUPS systems table. Because the database_backups system table contains historical information, it is not updated when you delete the backups.

- Open the vbr log file to check the status of a backup. The log file resides on the node where you have run vbr, in the directory specified by the tempDir vbr configuration parameter. The default name of this directory is /tmp/vbr.

## List Backups with vbr

To list backups on the backup hosts, use vbr --task listbackup with a specific configuration file. The following example shows how you can list backups, using a full backup configuration file, bak.ini:

```bash
dbadmin@doch01 ~]$ vbr --task listbackup --config-file /home/dbadmin/bak.ini
```

The vbr output information includes the following:

- **backup** — The name of the generated backup. Vertica names the backup by combining the name of the vbr configuration file with a timestamp. Use the timestamp to identify an archive when you perform a restore.

- **backup_type** — The type of the backup, either full or object.

- **epoch** — The epoch when the backup was created.

- **objects** — The objects being backed up. For a full backup, this field is blank.

- **nodes (hosts)** — The hosts that received the backup and the database node names that provided the backups.

- **file_system_type** — The storage location file system of the Vertica hosts that comprise this backup, either Linux or HDFS. For information on backing up HDFS hosts, refer to Backing Up HDFS Storage Locations.

The following example shows a list of full backups of a three node cluster to a single backup host, bkhost.

<table>
<thead>
<tr>
<th>backup</th>
<th>backup_type</th>
<th>epoch</th>
<th>objects</th>
<th>nodes (hosts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bak_20160414_134452</td>
<td>full</td>
<td>749</td>
<td>v_vmart_node0001(bkhost), v_vmart_node0002</td>
<td></td>
</tr>
<tr>
<td>(bkhost), v_vmart_node0003(bkhost)</td>
<td>[Linux]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bak_20160413_174544</td>
<td>full</td>
<td>659</td>
<td>v_vmart_node0001(bkhost), v_vmart_node0002</td>
<td></td>
</tr>
<tr>
<td>(bkhost), v_vmart_node0003(bkhost)</td>
<td>[Linux]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note: The listbackup task fails if you attempt to view backups on a cluster without a database when the backups were made to local hosts using the [] shortcut. Vbr requires a database to provide the location of the mapped local host.

Viewing All Backups in a Location

You can use the --list-all parameter with the listbackup task to view a list of all the snapshots stored on the hosts and paths listed in the specified configuration file.

```
dbadmin@doca01 ~]$ vbr --task listbackup --list-all --config-file /home/dbadmin/Nightly.ini
```

The following example shows a --list-all task using the configuration file Nightly.ini. That configuration file references the hosts doca01, doca02, and doca03 and the path /vertica/backup. The output shows that these locations contain not just the backups created using Nightly, but also backups created using a configuration file called Weekly.ini.

<table>
<thead>
<tr>
<th>backup</th>
<th>file_system_type</th>
<th>backup_type</th>
<th>epoch</th>
<th>objects</th>
<th>nodes(hosts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weekly_20170508_183249 (doca01)</td>
<td>[Linux]</td>
<td>full</td>
<td>3449</td>
<td></td>
<td>vmart_1(doca01), vmart_2(doca01), vmart_3</td>
</tr>
<tr>
<td>Weekly_20170508_182816 (doca03)</td>
<td>[Linux]</td>
<td>full</td>
<td>2901</td>
<td></td>
<td>vmart_1(doca01), vmart_2(doca02), vmart_3</td>
</tr>
<tr>
<td>Weekly_20170508_182754 (doca03)</td>
<td>[Linux]</td>
<td>full</td>
<td>2649</td>
<td></td>
<td>vmart_1(doca01), vmart_2(doca02), vmart_3</td>
</tr>
<tr>
<td>Weekly_20170508_182982 (doca03)</td>
<td>[Linux]</td>
<td>object</td>
<td>1794</td>
<td>sales_schema</td>
<td>vmart_1(doca01), vmart_2(doca02), vmart_3</td>
</tr>
<tr>
<td>Weekly_20170508_182988 (doca03)</td>
<td>[Linux]</td>
<td>object</td>
<td>1469</td>
<td>sales_schema</td>
<td>vmart_1(doca01), vmart_2(doca02), vmart_3</td>
</tr>
<tr>
<td>Weekly_20170508_183249 (doca03)</td>
<td>[Linux]</td>
<td>object</td>
<td>173</td>
<td>sales_schema</td>
<td>vmart_1(doca01), vmart_2(doca02), vmart_3</td>
</tr>
</tbody>
</table>

You can also use the the --json and --list-output-file parameters with the listbackup task to output the same content in JSON delimited format to a display or to an output file. For more information, refer to VBR Utility Reference.

Query database_backups

Use the following query to list historical information about backups. The objects column lists which objects were included in object-level backups. Do not use the backup_timestamp value when restoring an archive. Instead, Use the values provided by vbr --task listbackup, when restoring an archive.

```
VMart=> SELECT * FROM v_monitor.database_backups;
- [ RECORD 1 ]----+----------------------------------------------
```
### Related Topics

- **DATABASE_BACKUPS**

### Estimating Log File Disk Requirements

When you run the `vbr--setupconfig` command to create the configuration file and configure advanced parameters, one of the parameters is `tempDir`. This parameter specifies the database host location where `vbr` writes its log files and some other temp files.
negligible size). The default location is the /tmp/vbr directory on each database host. You can change the default location by specifying a different path in the configuration file.

The temporary storage directory also contains local log files describing the progress, throughput, and any errors encountered for each node. Each time you run vbr, the script creates a separate log file, each named with a timestamp. When using default settings, the log file typically uses about 4KB of space per node per backup.

The vbr log files are not removed automatically, so you must delete older log files manually, as necessary.

### Checking Backup Integrity

Vertica can confirm the integrity of your backup files and the manifest that identifies them. By default, backup integrity checks output their results to the command line.

### Perform a Quick Check

A quick check gathers all backup metadata from the backup location specified in the configuration file and compares that metadata to the backup manifest. A quick check does not verify the objects themselves. Instead, this task outputs an exceptions list of any discrepancies between objects in the backup location and objects listed in the backup manifest.

Use the following form to perform quick check task:

```
vbr -t quick-check -c configfile.ini
```

For example:

```
vbr -t quick-check -c backupconfig.ini
```

### Perform a Full Check

A full check verifies all objects listed in the backup manifest against file system metadata. A full check includes the same steps as a quick check. You can include the optional --report-file parameter to output results to a delimited JSON file. This task outputs an exceptions list that identifies the following inconsistencies:
- Incomplete restore points
- Damaged restore points
- Missing backup files
- Unreferenced files

Use the following form to perform a full check task:

```bash
evbr -t full-check -c configfile.ini --report-file=path/filename
```

For example:

```bash
evbr -t full-check -c backupconfig.ini --report-file=logging/fullintegritycheck.json
```

## Repairing Backups

Vertica can reconstruct backup manifests and remove unneeded backup objects. You can include the optional `--report-file` parameter to output results to a delimited JSON file.

## Performing a Quick Repair

The `quick-repair` task rebuilds the backup manifest, based on the manifests contained in the backup location.

Use the following form to perform a quick repair task:

```bash
evbr -t quick-repair -c configfile.ini
```

## Performing Garbage Collection

The `collect-garbage` task rebuilds your backup manifest and deletes any backup objects that do not appear in the manifest. You can include the optional `--report-file` parameter to output results to a delimited JSON file.

Use the following form to perform a garbage collection task:
Removing Backups

You can remove existing backups and restore points using the `vbr` utility. When you use the `vbr` task, `remove`, Vertica updates the manifests affected by the removal and maintains their integrity. If the backup archive contains multiple restore points, removing one does not affect the others. When you remove the last restore point, Vertica removes the backup entirely.

Note: Vertica does not support removing backups through the file system.

To remove a backup or a restore point, use the `vbr` task `remove`. Specify this task in the following form:

```
vbr -t remove -c configfile.ini --archive timestamp
```

You can remove multiple restore points using the `archive` parameter. To obtain the timestamp for a particular restore point, use the `listbackup` task.

- To remove multiple restore points, use a comma separator:
  ```
  --archive="restore_point1, restore_point2"
  ```

- To remove an inclusive range of restore points, use a colon:
  ```
  --archive="restore_point1: restore_point2"
  ```

- To remove all restore points, specify an archive value of `all`:
  ```
  --archive="all"
  ```

The following example shows how you can remove a restore point from an existing backup.

```
[dbadmin@doc01 ~] $ vbr -t remove -c backup.ini --archive 20160414_134452
Removing restore points: 20160414_134452
Remove complete!
```

Upgrading Pre-7.2 Backups

In version 7.2 and later, Vertica no longer relies on hard links to perform backups. As a result, pre-7.2 backups are not compatible with later Vertica versions. To resolve this issue, `vbr`
includes the vbr task 7.2_upgrade. This task copies an existing pre-7.2 backup and creates a 7.2.x-compatible version of it.

**Note:** Vertica recommends that you run this task before performing the first backup of the upgraded database.

To upgrade a backup:

1. Specify the vbr task 7.2_upgrade in the following form:

   ```
   vbr -t 7.2_upgrade --old-config-file outdated-configfile.ini -c new-configfile.ini
   ```

2. Verify that the `snapshotName` parameter is the same in the old and new configuration files.

The new configuration file assigns new backup locations for the upgraded backup. This approach preserves the existing backup so you can continue to perform incremental backups on the upgraded backup. After the upgrade is complete, Vertica no longer requires the old configuration file.

### Using HardFile Link Local Backups

You can use the vbr utility `hardLinkLocal` option to create a full- or object-level backups with hard file links on a local database host.

Creating hard-link local backups can provide the following advantages over a remote host backup:

- **Speed** — Hard link local backups are significantly faster than a remote host backup. When backing up, vbr does not copy files if the backup directory exists on the same file system as the database catalog and data directories.

- **Reduced network activities** — The hard-link local backup minimizes network load because it does not require rsync to copy files to a remote backup host.

- **Less disk space** — The backup includes a copy of the catalog and hard file links. Therefore, the local backup uses significantly less disk space than a backup with copies of database data files. However, a hard-link local backup saves a full copy of the catalog each time you run vbr. Thus, the disk size increases with the catalog size over time.

Hard-link local backups can help you during experimental designs and development cycles. Database designers and developers can create hard-link local object backups of schemas and
tables on a regular schedule during design and development phases. If any new developments are unsuccessful, developers can restore one or more objects from the backup.

**Note:** Running `vbr` does not affect active database applications. The `vbr` utility supports creating backups while concurrently running applications that execute DML statements, including `COPY`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.

## Planning Hard-Link Local Backups

If you plan to use hard-link local backups as a standard site procedure, design your database and hardware configuration appropriately. Consider storing all of the data files on one file system per node. Such a configuration has the advantage of being automatically set up for hard-link local backups. However, be aware that using one file system per node to support hard-link local backups does preclude the use of external storage locations on separate file systems.

## Specifying Backup Directory Locations

The `backupDir` parameter of the configuration file specifies the location of the top-level backup directory. Hard-link local backups require that the backup directory be located on the same Linux file system as the database data and catalog files. The Linux operating system cannot create hard file links to another file system.

Do not create the hard-link local backup directory in a database data storage location. For example, as a best practice, the database data directory should not be at the top level of the file system, as it is in the following example:

```
/home/dbadmin/data/VMart/v_vmart_node0001
```

Instead, Vertica recommends adding another subdirectory for data above the database level, such as in this example:

```
/home/dbadmin/data/dbdata/VMart/v_vmart_node0001
```

You can then create the hard-link local backups subdirectory as a peer of the data directory you just created, such as in this example:

```
/home/dbadmin/data/backups
/home/dbadmin/data/dbdata
```
Hard-Link Local Backups in Disaster Recovery

Hard-link local backups are only as reliable as the disk on which they are stored. If the local disk becomes corrupt, so does the hard link local backup. In this case, you are unable to restore the database from the hard-link local backup because it is also corrupt.

All sites should maintain full backups externally for disaster recovery because hard-link local backups do not actually copy any database files.

Related Tasks

- Creating Hard-Link Local Backups

Configuring Hard-Link Local Backup Hosts

When specifying the backupHost parameter for your hard link local configuration files, use the database host names (or IP addresses) as known to admintools. Do not use the node names. Host names (or IP addresses) are what you used when setting up the cluster. Do not use localhost for the backupHost parameter.

Listing Host Names

To query node names and host names:

```
VMart=> select node_name, host_name from node_resources;
    node_name | host_name
-------------+-----------
  v_vmart_node0001 | 192.168.223.11
  v_vmart_node0002 | 192.168.223.22
  v_vmart_node0003 | 192.168.223.33
(3 rows)
```

Because you are creating a local backup, use square brackets [ ] to map the host to the local host. For more information, refer to [Mapping].

```
[Mapping]
v_vmart_node0001 = [:]/home/dbadmin/data/backups
v_vmart_node0002 = [:]/home/dbadmin/data/backups
v_vmart_node0003 = [:]/home/dbadmin/data/backups
```
Restoring Object-Level Backups

To restore an object-level backup to the database from which it was taken, the database must be UP. The vbr configuration file you use for the restore task specifies the backup to restore.

You can specify how Vertica reacts to duplicate objects by configuring the objectRestoreMode setting in your vbr configuration file.

Restoring Objects to a Changed Cluster

Unlike restoring from a full database backup, vbr supports restoring object-level backups after adding nodes to the cluster. Any nodes that were not in the cluster when you created the object-level backup do not participate in the restore. You can rebalance your cluster after the restore to distribute data among the new nodes.

You cannot restore an object-level backup after removing nodes, altering node names, or changing IP addresses. Trying to restore an object-level backup after such changes causes vbr to fail and display this message:

Preparing...
Topology changed after backup; cannot restore.
restore failed!

Projection Epoch After Restore

All object-level backup and restore events are treated as DDL events. If a table does not participate in an object-level backup, possibly because a node being down, restoring the backup affects the projection in the following ways:

- Its epoch is reset to 0.
- It must recover any data that it does not have by comparing epochs and other recovery procedures.

Catalog Locks During Backup Restore

As with other databases, Vertica transactions follow strict locking protocols to maintain data integrity.
When restoring an object-level backup into a cluster that is UP, Vertica begins by copying data and managing storage containers. If necessary, Vertica splits the containers. This process does not require any database locks.

After completing data copying tasks, Vertica first requires a table object lock (O-lock) and then a global catalog lock (GCLX).

Sometimes, other database operations, such as DML statements, are in progress when the process attempts to get an O-lock on the table. In such cases, Vertica is blocked from progress until the DML statement completes and releases the lock. After securing an O-lock first, and then a GCLX lock, Vertica blocks other operations that require a lock on the same table.

To guarantee catalog consistency, processes can hold a GCLX for a minimal duration. When the restore locks are in effect, any concurrent table modifications are blocked until both locks are released. Database system operations, such as the Tuple Mover (TM) transferring data from memory to disk, are canceled to permit the object-level restore to complete.

Catalog Restore Events

Each object-level backup includes a section of the database catalog, called a snippet. A snippet contains the selected objects, their dependent objects, and principal objects. A catalog snippet is similar in structure to the database catalog but consists of a subset representing the object information. Objects being restored can be read from the catalog snippet and used to update both global and local catalogs.

Each object from a restored backup is updated in the catalog. If the object no longer exists, Vertica drops the object from the catalog. Any dependent objects that are not in the backup are also dropped from the catalog.

The Vertica utility uses existing dependency verification methods to check the catalog and adds a restore event to the catalog for each restored table. That event also includes the epoch at which the event occurred. If a node misses the restore table event, it recovers projections anchored on the given table.

Restore and Replicate Objects to a Newer Version of Vertica

Beginning with version 9.0.0-2, Vertica supports object replication and restoration to a target database up to one minor version later than the current database version. For example, you can replicate or restore objects from a version 9.0.0-2 database to a version 9.0.1 database. The restore or replication process from one version to another is the same as it is to the same version.
If your restored or replicated objects require a UDx library that is not present in the later version of your database, Vertica displays the following error:

```
ERROR 2858: Could not find function definition
```

You can resolve this issue by installing compatible libraries in your target database.

**Catalog Size Limitations**

Object level restores can fail if your catalog size is greater than five percent of the total memory available in the node performing the restore. In this situation, Vertica recommends restoring individual objects from the backup. For more information, refer to Restoring Individual Objects from a Full or Object-Level Backup.

**See Also**

- Failure Recovery
- Transactions

**Related Tasks**

- Creating Object-Level Backups
- Restoring Full Database Backups
- Ownership of Restored Objects

**Restoring Full Database Backups**

To restore a full database backup, you must verify that:

- The database is DOWN. You cannot restore a full backup when the database is running.
- All of the backup hosts are UP and available.
- The backup directory exists and contains the backups from which to restore the data.
The cluster to which you are restoring the backup has:
- The same number of hosts as the one used to create the backup
- Identical node names

The target database must already exist on the cluster to which you are restoring data.
- Database can be completely empty, without any data or schema.
- The database name must match the name in the backup
- All of the node names in the database must match the names of the nodes in the configuration file.

The user performing the restore is the database administrator.

You can use only a full database backup to restore a complete database. If you have saved multiple backup archives, you can restore from either the last backup or a specific archive. Restoring from a full database backup injects the OIDs from each backup into the restored catalog of the full database backup. The catalog also receives all archives. Additionally, the OID generator and current epoch are set to the current epoch.

### Restoring the Most-Recent Backup

To perform a full database restore, the cluster must be DOWN. Usually, when a node or cluster is DOWN, you want to return the cluster to its most-recent state. Doing so requires restoring a full database backup. You can restore any full database backup from the archive by identifying the name in the configuration file.

To restore from the most recent backup, use the configuration file used to create the backup, specifying `vbr` with the `--task restore`. If your password configuration file does not contain the database superuser password, the utility prompts you to enter it at run time.

The following example shows how you can use the `db.ini` configuration file for restoration:

```
> vbr --task restore --config-file db.ini
Copying...
1871652633 out of 1871652633, 100%
All child processes terminated successfully.
restore done!
```

You can restore a backup only to the database from which it was taken.
Restoring an Archive

If you saved multiple backups, you can specify a specific archive to restore. To list the archives that exist to choose one to restore, use the `vbr --listbackup` task, with a specific configuration file.

To restore from one of several archives:

Log in using the database administrator's account.

Invoke the utility with the `--task restore` command:

- Include the configuration file with which you created the backup.
- Next, add the file name with the `--archive` parameter with the `date_timestamp` suffix of the directory name, which identifies the archive to restore.

For example:

```
> vbr --task restore --config-file fullbak.ini --archive=20121111_205841
```

The `vbr` utility restores the backup.

The `--archive` parameter identifies the archive created on 11-11-2012 (`archive20121111`), at time `205841` (20:58:41). You need specify only the `_archive` suffix, because the configuration file identifies the backup name of the subdirectory, and the OID identifier indicates the backup is an archive.

See Also

- [Viewing Backups](#)
- [Restoring Individual Objects from a Full or Partial Backup](#)

Related Terms

- [database administrator](#)
Full Backup Restore to an Alternate Cluster

Vertica supports restoring a full backup to an alternate cluster.

Requirements for Restoring to an Alternate Cluster

The process is similar to the process for Restoring Full Database Backups, with the following additional requirements.

The destination database must:

- Be DOWN.
- Share the same name as the source database.
- Have the same number of nodes as the source database.
- Have the same names as the source nodes.
- Use the same catalog directory location as the source database.
- Use the same port numbers as the source database.
- Contain the same standby nodes as the backup.

Note: [[[Undefined variable Vertica.DBMS_UPPERCASE]]] does not support restoring a backup from an Enterprise cluster to an AWS cluster.

Restore a Full Backup to an Alternate Cluster

1. Copy the vbr configuration file that you used to create the backup to any node on the destination cluster.

2. If you are using a stored password, copy the password configuration file to the same location as the vbr configuration file.

3. From the destination node, issue a vbr restore command, such as:

   $ vbr -t restore -c full.ini
4. After the restore has completed, **start the restored database.**

**Restoring Individual Objects from a Full or Object-Level Backup**

You can copy individual tables or schemas from any backup that contains those objects without restoring the entire backup. This option is useful if you only need to restore a few objects and want to avoid the overhead of a larger scale restore. Your database must be running, and your nodes must be UP to restore individual objects.

When you restore an object, Vertica does not automatically restore any dependent objects. For example, if you restore a schema containing views, Vertica does not automatically restore the tables corresponding to those views.

You can specify how Vertica reacts to duplicate objects by configuring the `objectRestoreMode` setting in your `vbr` configuration file.

**Restoring an Object from the Most-Recent Backup**

To restore individual objects from the most recent backup, you use the `vbr` utility, just as you did to create the backup. Execute the command using the following form:

```
vbr --task=restore --config-file=filename --restore-objects=objectname,objectname
```

To restore an object, log in using the database administrator's account. You cannot run the utility as root.

To restore from the most recent backup, use the configuration file used to create the backup, specifying the restore `vbr` task. Be sure to include the `--restore-objects` parameter and the names of the objects that you want to restore. Identify tables by fully qualified `schema.name`. To restore multiple objects, use a comma as a delimiter. Alternatively, you can use wildcard patterns to identify objects to restore.

If your configuration file does not contain the database superuser password, the `vbr` utility prompts you to enter it at run time.

The following example uses the `db.ini` configuration file, which includes the database administrator's password:
Restoring an Object from an Archive

To restore individual objects from a specific archive, you use the same vbr utility that you used to create the backup. The command takes the following form:

```bash
> vbr --task=restore --config-file=filename --archive=date_time --restore-objects=objectname,objectname
```

Invoke the utility with the `--task restore` command, the configuration file with which you created the backup. Next, add the `--archive` parameter with the `date_timestamp` suffix of the directory name to identify which archive to restore.

The `--archive` parameter identifies the archive subdirectory, including date and time. For example, `_archive20121111_205841` was created on date 11-11-2012 at time 20:58:41. You need specify only the `date_time` suffix because the configuration .ini file identifies the backup name of the subdirectory. Additionally, the OID identifier indicates the backup is an archive.

For example:

```bash
> vbr --task=restore --config-file=fullbak.ini --archive=20121111_205841 --restore-objects=salesdb.contacts
```

The `vbr` utility restores the backup.

Restore and Replicate Objects to a Newer Version of Vertica

Beginning with version 9.0.0-2, Vertica supports object replication and restoration to a target database up to one minor version later than the current database version. For example, you
can replicate or restore objects from a version 9.0.0-2 database to a version 9.0.1 database. The restore or replication process from one version to another is the same as it is to the same version.

If your restored or replicated objects require a UDx library that is not present in the later version of your database, Vertica displays the following error:

```
ERROR 2858: Could not find function definition
```

You can resolve this issue by installing compatible libraries in your target database.

See Also

- Monitoring Recovery
- Viewing Backups
- Restoring Full Database Backups
- Restoring Object-Level Backups
- Ownership of Restored Objects
- Using Wildcards with Backup, Restore, and Replicate

**Restoring Hard-Link Local Backups**

Before attempting a hard-link local backup, you must be aware of issues around restoring from this type of backup, whether full or object level.

**Process for Restoring Full and Object-Level Hard Link Local Backups**

If you have created both full and object-level backups and the database fails, restore the backups in this order:

1. Restore the full database backup.
2. Restore from any object-level backups.
Transferring Backups to and from Remote Storage

When a full hard-link local backup exists, you can use a utility (other than vbr) to transfer the backup to another storage media, such as tape. Transferring hard-link local backups to another storage media may copy the data files associated with the hard file links.

You can use a different directory when you return the backup files to the hard-link local backup host. However, you must also change the backupDir parameter value in the configuration file before restoring the backup.

Complete the following steps to restore hard link local backups from external media:

1. If the original backup directory no longer exists on one or more local backup host nodes, re-create the directory.

   The directory structure into which you restore hard link backup files must be identical to what existed when the backup was created. For example, if you created hard-link local backups at the following backup directory, you can then re-create that directory structure:

   ```
   /home/dbadmin/backups/localbak
   ```

2. Copy the backup files to their original backup directory, as specified for each node in the configuration file. For more information, refer to [Mapping].

3. Restore the backup, using one of three options:

   a. To restore the latest version of the backup, move the backup files to the following directory:

   ```
   /home/dbadmin/backups/localbak/node_name/snapshotname
   ```

   b. To restore a different backup version, move the backup files to this directory:

   ```
   /home/dbadmin/backups/localbak/node_name/snapshotname_archivedate_timestamp
   ```

4. When the backup files are returned to their original backup directory, use the original configuration file to invoke vbr as follows:

   ```
   >/opt/vertica/bin/vbr --task restore --config-file localbak.ini
   ```

5. Confirm that the backup has succeeded. Verify that:
• The physical files are restored from tape into the correct directories.

• You are using the configuration file that specifies hardLinkLocal = true.

Note: You can use a different directory when you return the backup files to the hard-link local backup host. However, you must also change the backupDir parameter value in the configuration file before restoring the backup.

Ownership of Restored Objects

When performing a restore, Vertica inserts data into existing database objects. By default, the restore does not affect the ownership, storage policies, or permissions of the restored objects. If, however, the restored object does not currently exist, Vertica re-creates it. In this situation, the restored object is owned by the user (DBADMIN) performing the restore. Vertica does not restore dependent grants, roles, or client authentications with restored objects.

If the storage policies of a restored object are not valid, vbr applies the default storage policy. Restored storage policies can become invalid due to HDFS storage locations, table incompatibility, and unavailable min-max values at restore time.

Sometimes, Vertica encounters a catalog object that it does not need to restore. When this situation occurs Vertica generates a warning message for that object and the restore continues.

Examples

Suppose you have a full backup, including Schema1, owned by the user, Alice. Schema1 contains Table1, owned by Bob, who eventually passes ownership to Chris. The user dbadmin performs the restore. The following scenarios might occur that affect ownership of the schema:

Scenario 1:
Schema1.Table1 has been dropped at some point since the backup was created. When dbadmin performs the restore, Vertica re-creates Schema1.Table1. As the user performing the restore, dbadmin takes ownership of Schema1.Table1. Because Schema1 still exists, Alice retains ownership of the schema.

Scenario 2:
Schema1 is dropped, along with all contained objects. When dbadmin performs the restore, Vertica re-creates the schema and all contained objects. Dbadmin takes ownership of Schema1 and Schema1.Table1.

**Scenario 3:**

Schema1 and Schema1.Table1 both exist in the current database. When the user, dbadmin, rolls back to an earlier backup, the ownership of the objects remains unchanged. Alice owns Schema1 and Bob owns Schema1.Table1.

**Scenario 4:**

Schema1.Table1 exists and dbadmin wants to roll back to an earlier version. In the time since the backup was made, ownership of Schema1.Table1 has changed to Chris. When dbadmin restores Schema1.Table1, Alice remains owner of Schema1 and Chris remains owner of Schema1.Table1. The restore does not revert ownership of Schema1.Table1 from Chris to Bob.

**Restoring Data When Node Type Is Not Permanent**

The only node type on which Vertica supports data restoration is Permanent. You cannot restore data on the remaining node types:

- Ephemeral
- Execute
- Standby

To restore or replicate to these nodes, you must first change the destination node type to PERMANENT.

For more information, refer to Setting Node Type.

**Copying the Database to Another Cluster**

You can use the vbr utility to copy the entire database to another Vertica cluster. This feature helps you perform tasks such as copying a database between a development and a production environment. Copying your database to another cluster is essentially a simultaneous backup and restore operation. The data is backed up from the source database cluster and restored to the destination cluster in a single operation.
Notice: The copycluster task is not compatible with HDFS Storage Locations. Copycluster uses the Linux rsync command to copy files from the source cluster to the target cluster. HDFS storage backup and restore is based on use of snapshots. Data in an HDFS storage location is backed up to HDFS itself. Vertica cannot transfer data to a remote HDFS cluster the same way that it can for a Linux cluster.

The directory locations for the Vertica catalog, data, and temp directories must be identical on the source and target database. Use the following vsql query to view the source database directory locations. This example sets expanded display, for illustrative purposes, and lists the columns of most interest, node_name, storage_path, and storage_usage.

```
VMart=> \x
Expanded display is on.
VMart=> select node_name,storage_path, storage_usage from disk_storage;
- [ RECORD 1 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | /home/dbadmin/VMart/v_vmart_node0001_catalog/Catalog
storage_usage | CATALOG
- [ RECORD 2 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | /home/dbadmin/VMart/v_vmart_node0001_data
storage_usage | DATA,TEMP
- [ RECORD 3 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | home/dbadmin/SSD/schemas
storage_usage | DATA
- [ RECORD 4 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | /home/dbadmin/SSD/tables
storage_usage | DATA
- [ RECORD 5 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | /home/dbadmin/SSD/schemas
storage_usage | DATA
- [ RECORD 6 ]-+----------------------------------+-
node_name | v_vmart_node0001
storage_path | /home/dbadmin/VMart/v_vmart_node0002_catalog/Catalog
storage_usage | CATALOG
- [ RECORD 7 ]-+----------------------------------+-
node_name | v_vmart_node0002
storage_path | /home/dbadmin/VMart/v_vmart_node0002_data
storage_usage | DATA,TEMP
- [ RECORD 8 ]-+----------------------------------+-
node_name | v_vmart_node0002
storage_path | /home/dbadmin/SSD/tables
storage_usage | DATA
```

Notice the directory paths for the Catalog, Data, and Temp storage. These paths are the same on all nodes in the source database and must be the same in the target database.
Note: When copying a database to another cluster, if the target data is different, `vbr` overwrites all existing data. To retain existing data on the target cluster, create a full database backup of the target before invoking the `copycluster vbr` task.

Identifying Node Names for Target Cluster

Before you can configure the target cluster, you need to know the exact names that `admintools` supplied to all nodes in the source database.

To see the node names, run a query such as:

```
VMart=> select node_name from nodes;
    node_name
---------------
   v_vmart_node001
   v_vmart_node002
   v_vmart_node003
(3 rows)
```

You can also find the node names by running `Admintools` from the command line. For example, for the VMart database, you can enter a command such as:

```
$ /opt/vertica/bin/admintools -t node_map -d VMART
 DATABASE | NODENAME | HOSTNAME
-----------|----------|-------------------
 VMART     | v_vmart_node001 | 192.168.223.xx
 VMART     | v_vmart_node002 | 192.168.223.yy
 VMART     | v_vmart_node003 | 192.168.223.zz
```

Configuring the Target Cluster

Configure the target to allow the source database to connect to it and restore the database. The target cluster must:

- Have the same number of nodes the source cluster.
- Have a database with the same name as the source database. The target database can be completely empty.
- Have the same node names as the source cluster. The nodes names listed in the NODES system tables on both clusters must match.
- Be accessible from the source cluster.
- Have the same database administrator account, and all nodes must allow a database administrator of the source cluster to login through SSH without a password.

  Note: Passwordless access *within* the cluster is not the same as passwordless access *between* clusters. The SSH ID of the administrator account on the source cluster and the target cluster are likely not the same. You must configure each host in the target cluster to accept the SSH authentication of the source cluster.

- Have adequate disk space for the `vbr --task copycluster` command to complete.

Creating a Configuration File for CopyCluster

You must create a configuration file specifically for copying your database to another cluster. In the configuration file, specify the host names of nodes in the target cluster as the backup hosts. When you use the `copycluster` command, the `vbr` requires that you define the `backupHost`. However, the `vbr` ignores the `backupDir` option, and always stores the data in the catalog and data directories of the target database.

You cannot use an object-level backup with the `copycluster` command. Instead, you must perform a full database backup.

The following example shows how you can set up the `vbr` to copy a database on a 3-node cluster, `v_vmart`, to another cluster, `test-host`.

```plaintext
[Misc]
snapshotName = CopyVmart
restorePointLimit = 5
tempDir = /tmp/vbr
retryCount = 5
retryDelay = 1

[Database]
dbName = vmart
dbUser = dbadmin
dbPassword = password
dbPromptForPassword = False

[Transmission]
encrypt = False
checksum = False
port_rsync = 50000

[Mapping]; backupDir is not used for cluster copy
v_vmart_node0001= test-host01
v_vmart_node0002= test-host02
v_vmart_node0003= test-host03
```
Copying the Database

You must stop the target cluster before you invoke `copycluster`.

To copy the cluster, run `vbr` from a node in the source database using the database administrator account. The following example demonstrates how you can copy a cluster using a configuration file located in the current directory.

```
$ vbr -t copycluster -c copycluster.ini
Starting copy of database VMART.
Participating nodes: vmart_node0001, vmart_node0002, vmart_node0003, vmart_node0004.
Enter vertica password:
Snapshotting database.
Snapshot complete.
Determining what data to copy.
[==================================] 100%
Approximate bytes to copy: 987394852 of 987394852 total.
Syncing data to destination cluster.
[==================================] 100%
Reinitializing destination catalog.
Copycluster complete!
```

If the copy cluster task is interrupted, the destination cluster retains any data files that have already transferred. If you attempt the operation again, Vertica does not need to resend these files.

Related Tasks

- Configuring Backup Hosts

Replicating Tables and Schemas to an Alternate Database

You can replicate Vertica tables and schemas from one database to alternate databases in your organization. Using this strategy helps you:

- Replicate objects to a secondary site.
- Copy tables and schemas between test, staging, and production clusters that have the same number of database nodes.
Important: The vbr replicate task does not support copying an entire database; use the copycluster task to replicate an entire database.

Advantages of Alternate Database Replication

Replicating objects is generally faster than exporting and importing objects. As with Vertica backup and restore, object replication is incremental. The first replication of an object replicates the entire object. Subsequent replications copy only data that has changed since the last replication. Vertica replicates data as of the current epoch on the target database. Used with a cron job, you can replicate key objects to create a backup database.

In situations where the target database is down, or you plan to replicate the entire database, Vertica recommends that you try Copying the Database to Another Cluster.

Catalog Size Limitations

Object level restores can fail if your catalog size is greater than five percent of the total memory available in the node performing the restore. In this situation, Vertica recommends restoring individual objects from the backup. For more information, refer to Restoring Individual Objects from a Full or Object-Level Backup.

How DOWN Nodes Affect Replication

The effect of DOWN nodes on a replication task depends on whether they are present in the source or destination database.

<table>
<thead>
<tr>
<th>Location</th>
<th>Effect on Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWN source nodes</td>
<td>Vertica can replicate objects from a source database containing DOWN nodes. If nodes in the source database are DOWN, set the corresponding nodes in the target database to DOWN as well.</td>
</tr>
<tr>
<td>DOWN destination nodes</td>
<td>Vertica can replicate objects when the destination database has DOWN nodes. If nodes in the destination database are DOWN, you must exclude the corresponding source database nodes using the vbr --nodes parameter.</td>
</tr>
</tbody>
</table>
Replication to Alternate Database Process Flow

To replicate objects to an alternate database, begin in the SOURCE database and complete the following process:

1. Verify Replication Requirements
2. Edit Your vbr Configuration File
3. Replicate Objects
4. Monitor Object Replication

Verify Replication Requirements

To replicate objects to an alternate database, verify that your source and target databases meet the following requirements.

Both the source and target databases must:

- Have the same Linux user associated with the dbadmin account on both the source and target databases.
- Be UP.
- Have the same number of nodes.
- All nodes allowing a database administrator of the source cluster to login through SSH without a password.

Note: Passwordless access within the cluster is not the same as passwordless access between clusters. The SSH ID of the administrator account on the source cluster and the target cluster are likely not the same. You must configure each host in the target cluster to accept the SSH authentication of the source cluster.

Edit Your vbr Configuration File for Replication

The default configuration file that vbr creates does not contain all of the settings necessary to replicate objects to an alternate cluster.

Add the following parameters to the configuration file that you are using to replicate objects:
1. In the [Misc] section of your vbr configuration file, add the following parameter:

```
; Identify the objects that you want to replicate
objects = schema.objectName
```

2. In the [Database] section of your vbr configuration file, add the following parameters:

```
; parameters used to replicate objects between databases
dest_dbName =
dest_dbUser =
dest_dbPromptForPassword =
```

If you are using a stored password, be sure to configure the dest_dbPassword parameter in your Password Configuration File Reference.

3. In the [Mapping] section, map your source nodes to your target hosts:

```
[Mapping]
v_source_node0001 = targethost01
v_source_node0002 = targethost02
v_source_node0003 = targethost03
```

Replicate Objects

To replicate objects, use the vbr task replicate. Specify this task in the following form:

```
vbr -t replicate -c configfile.ini
```

Restore and Replicate Objects to a Newer Version of Vertica

Beginning with version 9.0.0-2, Vertica supports object replication and restoration to a target database up to one minor version later than the current database version. For example, you can replicate or restore objects from a version 9.0.0-2 database to a version 9.0.1 database. The restore or replication process from one version to another is the same as it is to the same version.

If your restored or replicated objects require a UDx library that is not present in the later version of your database, Vertica displays the following error:

```
ERROR 2858: Could not find function definition
```
You can resolve this issue by installing compatible libraries in your target database.

Monitor Object Replication

You can monitor object replication the following ways:

- Viewing `vbr` logs on the source database
- Checking database logs on the source and destination databases
- Querying `REMOTE_REPLICATION_STATUS` on the source database

Troubleshooting Backup and Restore

These tips can help you avoid issues related to backup and restore with Vertica and to troubleshoot any problems that occur.

Object Replication Fails

Confirm that you have excluded all DOWN nodes from your object replication.

If you do not exclude the DOWN node, replication fails with the following error:

Error connecting to a destination database node on the host `<hostname>` : `<error>` ...
Recovering the Database

Recovering a database can consist of any of the following:

- Restarting Vertica on a host
- Restarting the Database
- Recovering the Cluster From a Backup
- Replacing Failed Disks
- Copying the Database to Another Cluster
- Exporting a Catalog for support purposes
- Recovering a node on a per-table basis

You can monitor a recovery in progress by viewing log messages posted to the `vertica.log` file on each host.

See Also

- Failure Recovery

Failure Recovery

Vertica can restore the database to a fully functional state after one or more nodes in the system experiences a software- or hardware-related failure. Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure can cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, queries other nodes in the cluster to recover lost objects and catch up with database changes.

K-safety sets fault tolerance for the database cluster, where K can be set to 0, 1, or 2. The value of K specifies how many copies Vertica creates of segmented projection data. If K-safety for a database is set to 1 or 2, Vertica creates K+1 instances, or buddies, of each projection segment. Vertica distributes these buddies across the database cluster, such that projection data is protected in the event of node failure. If any node fails, the database can continue to process
queries so long as buddies of data on the failed node remain available elsewhere on the cluster.

Note: You can monitor the cluster state through the View Database Cluster state menu option.

Recovery Scenarios
Vertica begins the database recovery process when you restart failed nodes or the database. The mode of recovery for a K-safe database depends on the type of failure:

- One or more nodes in the database failed, but the database continued to operate.
- The database shut down cleanly.
- The database shut down uncleanly.

In the first two cases, node recovery is automatic; in the third case (unclean shutdown), recovery requires manual intervention by the database administrator. The following sections discuss these cases in greater detail.

Recovery of failed nodes
One or more nodes failed but the remaining nodes in the database filled in for them, so database operations continued without interruption. Use Administration Tools to restart failed nodes through the Restart Vertica on Host option. While restarted nodes recover their data from other nodes, their status is set to RECOVERING. Except for a short period at the end, the recovery phase has no effect on database transaction processing. After recovery is complete, the restarted nodes status changes to UP.

Recovery after clean shutdown
The database was shut down cleanly through Administration Tools. To restart the database, use the Start Database option. On restart, all nodes whose status was UP before the shutdown resume a status of UP. If the database contained one or more failed nodes on shutdown and they are now available, they begin the recovery process as described above.

Recovery after unclean shutdown
Reasons for unclean shutdown include:

- A critical node failed, leaving part of the database's data unavailable.
- A site-wide event such as a power failure causes all nodes to reboot.
- Vertica processes on the nodes exited due to a software or hardware failure.
Unclean shutdown can put the database in an inconsistent state—for example, Vertica might have been in the middle of writing data from WOS to disk at the time of failure, and this process was left incomplete. When you restart the database through the Administration Tools, Vertica determines that normal startup is not possible and uses the Last Good Epoch to determine when data was last consistent on all nodes. When you restart the database, Vertica prompts you to accept recovery with the suggested epoch. If accepted, the database recovers and all data changes after the Last Good Epoch are lost. If not accepted, startup is aborted.

Instead of accepting the recommended epoch, you can recover from a backup. You can also choose an epoch that precedes the Last Good Epoch, through the Administration Tools Advanced Menu option Roll Back Database to Last Good Epoch. This is useful in special situations—for example the failure occurs during a batch of loads, where it is easier to restart the entire batch, even though some of the work must be repeated. In most cases, you should accept the recommended epoch.

### Epochs and Node Recovery

The checkpoint epoch (CPE) for both the source and target projections are updated as ROS containers are moved. The start and end epochs of all storage containers, such as ROS containers, are modified to the commit epoch. When this occurs, the epochs of all columns without an actual data file rewrite advance the CPE to the commit epoch of move_partitions_to_table. If any nodes are down during the TM moveout and/or during the move partition, they will detect that there is storage to recover, and will recover from other nodes with the correct epoch upon rejoining the cluster.

### Manual Recovery Notes

- You can manually recover a database where up to K nodes are offline—for example, they were physically removed for repair or not reachable at the time of recovery. When the missing nodes are restored, they recover and rejoin the cluster as described earlier in Recovery Scenarios.

- You can manually recover a database if the nodes to be restarted can supply all partition segments, even if more than K nodes remain down at startup. In this case, all data is available from the remaining cluster nodes, so the database can successfully start.
The default setting for the HistoryRetentionTime configuration parameter is 0, so Vertica only keeps historical data when nodes are down. This setting prevents use of the Administration Tools Roll Back Database to Last Good Epoch option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch that precedes the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window to remove loaded data—for example:

```sql
=> ALTER DATABASE mydb SET HistoryRetentionTime = 86400;
```

See Epoch Management Parameters in the Administrator's Guide.

When a node is down and manual recovery is required, it can take a full minute or longer for Vertica processes to time out while the system tries to form a cluster. Wait approximately one minute until the system returns the manual recovery prompt. Do not press CTRL-C during database startup.

See Also

High Availability

## Restarting Vertica on a Host

When one node in a running database cluster fails, or if any files from the catalog or data directories are lost from any one of the nodes, you can check the status of failed nodes using either the Administration Tools or the Management Console.

### Restarting Vertica on a Host Using the Administration Tools

1. Run Administration Tools.

2. From the Main Menu, select Restart Vertica on Host and click OK.

3. Select the database host you want to recover and click OK.
4. Verify recovery state by selecting View Database Cluster State from the Main Menu.

After the database is fully recovered, you can check the status at any time by selecting View Database Cluster State from the Administration Tools Main Menu.

**Restarting Vertica on a Host Using the Management Console**

1. Connect to a cluster node (or the host on which MC is installed).
2. Open a browser and connect to MC as an MC administrator.
3. On the MC Home page, double-click the running database under the Recent Databases section.
4. Within the Overview page, look at the node status under the Database sub-section and see if all nodes are up. The status will indicate how many nodes are up, critical, down, recovering, or other.
5. If a node is down, click Manage at the bottom of the page and inspect the graph. A failed node will appear in red.
6. Click the failed node to select it and in the Node List, click the Start node button.

**Restarting the Database**

If you lose the Vertica process on more than one node (for example, due to power loss), or if the servers are shut down without properly shutting down the Vertica database first, the database cluster indicates that it did not shut down gracefully the next time you start it.

The database automatically detects when the cluster was last in a consistent state and then shuts down, at which point an administrator can restart it.

From the Main Menu in the Administration Tools:
1. Verify that the database has been stopped by clicking Stop Database.

A message displays: No databases owned by <dbadmin> are running

2. Start the database by selecting Start Database from the Main Menu.

3. Select the database you want to restart and click OK.

If you are starting the database after an unclean shutdown, messages display, which indicate that the startup failed. Press RETURN to continue with the recovery process.

An epoch represents committed changes to the data stored in a database between two specific points in time. When starting the database, Vertica searches for last good epoch.

4. Upon determining the last good epoch, you are prompted to verify that you want to start the database from the good epoch date. Select Yes to continue with the recovery.
Caution: If you do not want to start from the last good epoch, you may instead restore the data from a backup and attempt to restart the database. For this to be useful, the backup must be more current than the last good epoch.

Vertica continues to initialize and recover all data prior to the last good epoch.

If recovery takes more than a minute, you are prompted to answer <Yes> or <No> to "Do you want to continue waiting?"

When all the nodes' status have changed to RECOVERING or UP, selecting <No> lets you exit this screen and monitor progress via the Administration Tools Main Menu. Selecting <Yes> continues to display the database recovery window.

Note: Be sure to reload any data that was added after the last good epoch date to which you have recovered.
Recovering the Cluster From a Backup

To recover a cluster from a backup, refer to the following topics:

- Backing Up and Restoring the Database
- Restoring Full Database Backups

Monitoring Recovery

When your Vertica database is recovering from a failure, it's important to monitor the recovery process. There are several ways to monitor database recovery:

Phases of a Recovery

The phases of a Vertica recovery are the same regardless of whether you are recovering by table or node. In the case of a recovery by table, tables become individually available as they complete the final phase. In the case of a recovery by node, the database objects only become available after the entire node completes recovery.

When you perform a recovery in Vertica, each recovered table goes through the following phases:

<table>
<thead>
<tr>
<th>Order</th>
<th>Phase</th>
<th>Description</th>
<th>Lock Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Historical</td>
<td>Vertica copies any historical data it may have missed while in a state of DOWN or INITIALIZING.</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>Historical Dirty</td>
<td>Vertica recovers any DML transactions that committed after the node or table began recovery.</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>Current Replay Delete</td>
<td>Vertica replays any delete transactions that took place during the recovery.</td>
<td>T-lock</td>
</tr>
<tr>
<td>4</td>
<td>Aggregate Projections</td>
<td>Vertica recovers any aggregate projections.</td>
<td>T-lock</td>
</tr>
</tbody>
</table>
After a table completes the last phase, Vertica considers it fully recovered. At this point, the table can participate in DDL and DML operations.

**Exporting a Catalog**

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the `EXPORT_CATALOG` function in the SQL Reference Manual for details.

**Best Practices for Disaster Recovery**

To protect your database from site failures caused by catastrophic disasters, maintain an off-site replica of your database to provide a standby. In case of disaster, you can switch database users over to the standby database. The amount of data loss between a disaster and fail over to the offsite replica depends on how frequently you save a full database backup.

The solution to employ for disaster recover depends upon two factors that you must determine for your application:

- **Recovery point objective (RPO):** How much data loss can your organization tolerate upon a disaster recovery?

- **Recovery time objective (RTO):** How quickly do you need to recover the database following a disaster?

Depending on your RPO and RTO, Vertica recommends choosing from the following solutions:

1. **Dual-load:** During each load process for the database, simultaneously load a second database. You can achieve this easily with off-the-shelf ETL software.

2. **Periodic Incremental Backups:** Use the procedure described in Copying the Database to Another Cluster to periodically copy the data to the target database. Remember that the script copies only files that have changed.

3. **Replication solutions provided by Storage Vendors:** Although some users have had success with SAN storage, the number of vendors and possible configurations prevent Vertica from providing support for SANs.
The following table summarizes the RPO, RTO, and the pros and cons of each approach:

<table>
<thead>
<tr>
<th></th>
<th>Dual Load</th>
<th>Periodic Incremental</th>
<th>Storage Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RPO</strong></td>
<td>Up to the minute data</td>
<td>Up to the last backup</td>
<td>Recover to the minute</td>
</tr>
<tr>
<td><strong>RTO</strong></td>
<td>Available at all times</td>
<td>Available except when backup in progress</td>
<td>Available at all times</td>
</tr>
</tbody>
</table>
| **Pros**       | • Standby database can have different configuration  
                   • Can use the standby database for queries | • Built-in scripts                     | Transparent to the database           |
| **Cons**       | • Possibly incur additional ETL licenses  
                   • Requires application logic to handle errors | Need identical standby system           | • More expensive                      
                   • Media corruptions are also replicated |                                        |
In clusters with a failed node, the database optimizer chooses different segmentations of buddy projections to distribute the workload throughout the cluster. Query performance within a cluster containing a failed node can, however, deteriorate, as one node must now do the work of two nodes. The extent of the performance impact varies depending on the queries that you run and the location of the data that you are querying. Query performance remains affected until the failed node has been completely restored.

**Recovery By Table**

Vertica supports node recovery on a per-table basis. Unlike a node-based recovery, recovering by table makes tables available as they recover, before the node itself is completely restored. You can prioritize your most important tables to ensure that they become available as soon as possible. Recovered tables support all DDL and DML operations.

To enhance recovery speed, Vertica recovers multiple tables in parallel. The maximum number of tables recoverable at one time is governed by the MAXCONCURRENCY parameter in the RECOVERY resource pool.

Once a node has fully recovered, it enables full Vertica functionality.

Recovery by table is enabled by default.

**Note:** Vertica does not support recovery by table for tables with pre-join projections.

**Enabling Recovery by Table**

With all nodes in your cluster up, enable recovery by table with the following query:

```
SELECT SET_RECOVER_BY_TABLE('true');
```

**Important:** Do not disable recovery by table. SET_RECOVER_BY_TABLE has been deprecated and will be removed in a future release.

**Prioritizing Table Recovery**

You can specify the order in which Vertica recovers tables. This feature ensures that your most critical tables become available as soon as possible. To specify the recovery order of your tables, assign an integer priority value. Tables with higher priority values recover first. For
example, a table with a priority of 1000 is recovered before a table with a value of 500. Table priorities have the maximum value of a 64-bit integer.

If you do not assign a priority, or if multiple tables have the same priority, Vertica restores tables by OID order. Assign a priority with a query such as this:

```sql
=> SELECT set_table_recover_priority('avro_basic', '1000');
set_table_recover_priority
-----------------------------------------------
Table recovery priority has been set.
(1 row)
```

View assigned priorities with a query using this form:

```sql
SELECT table_name, recover_priority FROM v_catalog.tables;
```

The next example shows prioritized tables from the VMart sample database. In this case, the table with the highest recovery priorities are listed first (DESC). The shipping_dimension table has the highest priority and will be recovered first. (Example has hard Returns for display purposes.)

```sql
=> SELECT table_name AS Name, recover_priority from v_catalog.tables WHERE recover_priority > 1
   ORDER BY recover_priority DESC;
   Name   | recover_priority
---------|-------------------
shipping_dimension | 60000
warehouse_dimension | 50000
employee_dimension | 40000
vendor_dimension | 30000
date_dimension | 20000
promotion_dimension | 10000
iris2 | 9999
product_dimension | 10
customer_dimension | 10
(9 rows)
```

Viewing Table Recovery Status

View general information about a recovery querying the V_MONITOR.TABLE_RECOVERY_STATUS table. You can also view detailed information about the status of the recovery the table being restored by querying the V_MONITOR.TABLE_RECOVERIES table.
Collecting Database Statistics

The Vertica cost-based query optimizer relies on data statistics to produce query plans. If statistics are incomplete or out-of-date, the optimizer is liable to use a sub-optimal plan to execute a query.

For example, you load timestamp data into a large table at regular intervals and query the table for the mostly recently loaded rows. The following scenarios influence how optimizer chooses a plan:

- You load days 1 through 15 into the table and run ANALYZE_STATISTICS. When you next run a query that requests yesterday's data by filtering on the timestamp column, the optimizer chooses an optimized query plan.

- The next day, you load day 16 data and run the same query. If you do not run ANALYZE_STATISTICS again, the optimizer might conclude that the predicate results in only one row being returned because the date range falls outside the histogram range and the data becomes stale. When the optimizer detects that statistics are not current for a particular predicate—in this case, because a timestamp predicate is outside a histogram's boundary—Vertica uses other considerations to plan the query, such as FK-PK constraints.

In this case, you can run ANALYZE_STATISTICS after loading new data on day 16. You can also look for statistics in the EXPLAIN-generated query plan. For example, when statistics are outside a histogram's boundaries, the query plan is annotated with a status.

Vertica provides a number of tools like ANALYZE_STATISTICS for analyzing and updating database statistics. For an overview, see Database Statistics Tools.

Database Statistics Tools

Statistics collection is a cluster-wide operation that accesses data using a historical query (at epoch latest) without any locks. After statistics are computed, they are stored in the catalog and replicated on all nodes. The storage operation requires a brief, exclusive lock on the catalog, similar to when a DDL operation occurs. In fact, these operations require a COMMIT for the current transaction.

Vertica provides a number of tools for analyzing and updating database statistics:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE_</td>
<td>Collects a statistical data sampling.</td>
</tr>
<tr>
<td>Tool</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STATISTICS</td>
<td></td>
</tr>
<tr>
<td>ANALYZE_ROW_COUNT</td>
<td>Invoked through the Vertica function DO_TM_TASK, collects projection row counts.</td>
</tr>
<tr>
<td>ANALYZE_EXTERNAL_ROW_COUNT</td>
<td>Collects row count data for external tables.</td>
</tr>
<tr>
<td>EXPORT_STATISTICS</td>
<td>Generates database statistics in XML format from data previously collected by ANALYZE_STATISTICS.</td>
</tr>
<tr>
<td>PROJECTION_COLUMNS</td>
<td>Monitors information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated.</td>
</tr>
</tbody>
</table>

For more information about these analysis tools, see Analyzing Row Counts and Getting Statistics.

### See also

For information about tools to monitor and analyze query performance, see Analyzing Workloads.

For information about managing statistics, see descriptions of the following Vertica functions in the SQL Reference Manual:

- EXPORT_STATISTICS
- IMPORT_STATISTICS
- DROP_STATISTICS
- DROP_EXTERNAL_ROW_COUNT

### Analyzing Row Counts

Vertica lets you obtain row counts for projections and for external tables, through ANALYZE_ROW_COUNT and ANALYZE_EXTERNAL_ROW_COUNT, respectively.
Projection Row Count

ANALYZE_ROW_COUNT is a lightweight operation that automatically collects the number of rows in a projection every 60 seconds to collect a minimal set of statistics and aggregate row counts calculated during loads. You can invoke this function through the function DO_TM_TASK. For example:

```sql
=> SELECT DO_TM_TASK ('analyze_row_count', 'public.Emp_Dimension_b0');
```

To change the collection interval, set the configuration parameter AnalyzeRowCountInterval (see Configuration Parameters). For example, you can change the collection interval to 1 hour (3600 seconds) as follows:

```sql
=> ALTER DATABASE mydb SET AnalyzeRowCountInterval = 3600;
```

External Table Row Count

ANALYZE_EXTERNAL_ROW_COUNT calculates the exact number of rows in an external table. The optimizer uses this count to optimize for queries that access external tables. This is especially useful when an external table participates in a join. This function enables the optimizer to identify the smaller table to use as the inner input to the join, and facilitate better query performance.

The following query calculates the exact number of rows in the external table loader_rejects:

```sql
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('loader_rejects');
ANALYZE_EXTERNAL_ROW_COUNT
----------------------------
0
```

Getting Statistics

ANALYZE_STATISTICS collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table. You call this function as follows:

```sql
ANALYZE_STATISTICS ('[schema.]table'
  [, 'column[,....]'
  [, percent ]
)
If you call this function with a single empty string argument, Vertica collects statistics for all database tables and their projections:

```sql
=> SELECT ANALYZE_STATISTICS ('');
```

### Operations

ANALYZE_STATISTICS performs the following operations:

- Performs fast data sampling, which expedites analysis of relatively small tables with a large number of columns.
- Includes data from WOS.
- Recognizes deleted data instead of ignoring delete markers.
- Records in system table `PROJECTION_COLUMNS` the last time statistics were run for a table.
- Enables table analysis tables on a per-column basis for improved performance.

### Sampling Size

ANALYZE_STATISTICS constructs a column histogram from a set of rows that it randomly selects from all collected data. Regardless of the percentage setting, the function always creates a statistical sample that contains up to (approximately) the smaller of:

- $2^{17} (131,072)$ rows
- Number of rows that fit in 1 GB of memory

If a column has fewer rows than the maximum sample size, ANALYZE_STATISTICS reads all rows from disk and analyzes the entire column.

**Note:** The data collected in a sample range does not indicate how data should be distributed.

The following table shows how ANALYZE_STATISTICS, when set to different percentages, obtains a statistical sample from a given column:

<table>
<thead>
<tr>
<th>Number of column rows</th>
<th>%</th>
<th>Number of rows read</th>
<th>Number of sampled rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq \text{max-sample-size}$</td>
<td>20</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Number of column rows</td>
<td>%</td>
<td>Number of rows read</td>
<td>Number of sampled rows</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>---------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>400K</td>
<td>10</td>
<td>\textit{max-sample-size}</td>
<td>\textit{max-sample-size}</td>
</tr>
<tr>
<td>4000K</td>
<td>10</td>
<td>400K</td>
<td>\textit{max-sample-size}</td>
</tr>
</tbody>
</table>

Note: When a column specified for \texttt{ANALYZE\_STATISTICS} is first in a projection's sort order, the function reads all data from disk to avoid a biased sample.

Data Collection Percentage

If \texttt{ANALYZE\_STATISTICS} omits specifying a percentage, Vertica collects a fixed 10-percent sample of statistical data from disk. Specifying a percentage of data to read from disk gives you more control over deciding between sample accuracy and speed.

The percentage of data you collect affects collection time and accuracy:

- A smaller percentage is faster but returns a smaller data sample, which might compromise histogram accuracy.
- A larger percentage reads more data off disk. Data collection is slower, but a larger data sample enables greater histogram accuracy.

Evaluating Results

After you collect the desired statistics, run the Workload Analyzer to retrieve hints about under-performing queries and their root causes, and obtain tuning recommendations. For more information, see \textit{Analyzing Workloads}.

Examples

Compute statistics on all projections in the \texttt{VMart} database:

\[
\text{=> SELECT ANALYZE\_STATISTICS ('');}
\]

\texttt{ANALYZE\_STATISTICS}

\texttt{-------------}

\texttt{0}

\texttt{(1 row)}

Compute statistics on a single table:
Compute statistics on a single column across all projections for a table:

```sql
=> SELECT ANALYZE_STATISTICS ('shipping_dimension');
ANALYZE_STATISTICS
---------------------
  0
(1 row)
```

Collect data from the entire disk by setting the percent parameter to 100:

```sql
=> SELECT ANALYZE_STATISTICS ('shipping_dimension', 'shipping_key', 100);
ANALYZE_STATISTICS
---------------------
  0
(1 row)
```

Collect data on all projections for `shipping_dimension` from 20 percent of the disk:

```sql
=> SELECT ANALYZE_STATISTICS ('shipping_dimension', 20);
ANALYZE_STATISTICS
---------------------
  0
(1 row)
```

---

**Reporting Statistics**

Vertica supplies hints about statistics two ways:

- The `EXPLAIN`-generated query plan is annotated with a status. See [Reacting to Stale Statistics](#).

- The last time `ANALYZE_STATISTICS` was run for a table is recorded, so that subsequent calls to the function are optimized. This is useful during the database design process because if Database Designer does not collect statistics when adding design tables, it generates a warning indicating that statistics are old. You can then decide whether to run `ANALYZE_STATISTICS` before you proceed with the design.

Two columns in the `V_CATALOG.PROJECTION_COLUMNS` system table capture statistical information, as follows:
• **STATISTICS_TYPE** returns the type of statistics the column contains, one of the following: NONE, ROWCOUNT, or FULL.

• **STATISTICSUPDATED_TIMESTAMP** returns the last time statistics were collected in this table.

## Determining When Statistics Were Last Updated

System table **PROJECTION_COLUMNS** returns statistics about projection columns, including the type of statistics and when they were last updated.

For example, the following sample schema defines a table named trades, which groups the highly-correlated columns bid and ask and stores the stock column separately:

```sql
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION trades_p (  
    stock ENCODING RLE, GROUPED(bid ENCODING DELTAVAL, ask)) 
AS (SELECT * FROM trades) ORDER BY stock, bid;
=> INSERT INTO trades VALUES('acme', 10, 20);
=> COMMIT;
```

Query the **PROJECTION_COLUMNS** table for table trades:

```sql
=> \x
Expanded display is on.
=> SELECT * FROM PROJECTION_COLUMNS WHERE table_name = 'trades';
```

The **statistics_type** column returns NONE for all three columns in the trades table. The **statistics_updated_timestamp** field is empty because statistics have not yet been run on this table.

<table>
<thead>
<tr>
<th>RECORD 1</th>
<th>projection_id</th>
<th>45035996273718838</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>projection_name</td>
<td>trades_p</td>
</tr>
<tr>
<td></td>
<td>projection_column_name</td>
<td>stock</td>
</tr>
<tr>
<td></td>
<td>column_position</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>sort_position</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>column_id</td>
<td>45035996273718840</td>
</tr>
<tr>
<td></td>
<td>data_type</td>
<td>char(5)</td>
</tr>
<tr>
<td></td>
<td>encoding_type</td>
<td>RLE</td>
</tr>
<tr>
<td></td>
<td>access_rank</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>group_id</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>table_schema</td>
<td>public</td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>45035996273718836</td>
</tr>
<tr>
<td></td>
<td>table_name</td>
<td>trades</td>
</tr>
<tr>
<td></td>
<td>table_column_id</td>
<td>45035996273718836-1</td>
</tr>
</tbody>
</table>
Now, run statistics on the `stock` column:

```
=> SELECT ANALYZE_STATISTICS('trades.stock');
```

The system returns 0 for success:
Now query PROJECTION_COLUMNS again:

```sql
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
```

This time, statistics_type changes to FULL for the trades.stock column (representing full statistics were run). The statistics_updated_timestamp column returns the time the stock columns statistics were updated. The timestamp for the bid and ask columns have not changed because statistics were not run on those columns. Also, the bid and ask columns changed from NONE to ROWCOUNT because Vertica automatically updates ROWCOUNT statistics from time to time. The statistics are created by looking at existing catalog metadata.
If you run statistics on the `bid` column and then query this system table again, only RECORD 2 is updated:

```sql
=> SELECT ANALYZE_STATISTICS('trades.bid');

=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
```
You can query the `statistics_updated_timestamp` column to see when columns were updated:

```sql
=> \x
Expanded display is off.
```
Reacting to Stale Statistics

During predicate selectivity estimation, the query optimizer can identify when histograms are not available or are out of date. If the value in the predicate is outside the histogram's maximum range, the statistics are stale. If no histograms are available, then no statistics are available to the plan.

When the optimizer detects stale or no statistics, such as when it encounters a column predicate for which it has no histogram, the optimizer performs the following actions:

- Displays and logs a message that you should run `ANALYZE_STATISTICS`.
- Annotates `EXPLAIN`-generated query plans with a statistics entry.
- Ignores stale statistics when it generates a query plan. The optimizer uses other considerations to create a query plan, such as FK-PK constraints.

For example, the following query plan fragment shows no statistics (histograms unavailable):

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The following query plan fragment shows that the predicate falls outside the histogram range:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 35, Rows: 1 (PREDICATE VALUE OUT-OF-RANGE)]
```

You can get information about which table column has no statistics by querying a system table; for example, view the timestamp for when statistics were last run by querying system table `PROJECTION_COLUMNS`.

```sql
=> SELECT ANALYZE_STATISTICS('trades');
ANALYZE_STATISTICS
-------------
0
(1 row)
=> SELECT projection_column_name, statistics_type, statistics_updated_timestamp
     FROM PROJECTION_COLUMNS where table_name = 'trades';
projection_column_name | statistics_type | statistics_updated_timestamp
------------------------|----------------|-----------------------------
stock                   | FULL           | 2012-12-08 13:54:27.428622-05
bid                     | FULL           | 2012-12-08 13:54:27.428632-05
ask                     | FULL           | 2012-12-08 13:54:27.428639-05
(3 rows)
```
Example

1. Run full statistics on table trades:

```sql
=> SELECT ANALYZE_STATISTICS('trades');
ANALYZE_STATISTICS

(1 row)
```

2. Query columns column_name, statistics_type, and statistics_updated_timestamp from projection PROJECTION_COLUMNS.

```sql
=> SELECT projection_column_name, statistics_type, statistics_updated_timestamp
FROM PROJECTION_COLUMNS where table_name = 'trades';

<table>
<thead>
<tr>
<th>projection_column_name</th>
<th>statistics_type</th>
<th>statistics_updated_timestamp</th>
</tr>
</thead>
</table>
```

(3 rows)

You can also query the PROJECTIONS column HAS_STATISTICS, which returns true only when all non-epoch columns for a table have full statistics. Otherwise, the column returns false.

See Also

Analyzing Workloads

Canceling Statistics Collection

To cancel statistics collection mid analysis, execute CTRL-C on vsql or call the INTERRUPT_STATEMENT() function.

If you want to remove statistics for the specified table or type, call the DROP_STATISTICS() function.

Caution: After you drop statistics, it can be time consuming to regenerate them.
Best Practices for Statistics Collection

When you query a table, the query optimizer requires representative statistics on its data in order to choose the best query plan. For many applications, statistics do not need to be accurate to the latest minute. Vertica regularly calls ANALYZE ROW COUNT, which collects partial statistics and supplies enough data to satisfy many optimizer requirements.

Requesting Statistics

You can explicitly request Vertica to gather detailed information on the data of individual tables and their columns, by calling ANALYZE_STATISTICS. This function analyzes distribution of column data and storage usage across all projections. Without this data, the query optimizer assumes uniform distribution of data values and equal storage usage for all projections, and creates query plans accordingly.

You can specify the size of the data sample—by default, 10 percent. You can also narrow the scope of the collection by specifying individual columns.

You should call ANALYZE_STATISTICS on a table or individual columns when one or more of following conditions are true:

- Data is bulk loaded for the first time.
- A new projection is refreshed.
- The number of rows changes significantly.
- A new column is added to the table.
- Column minimum/maximum values change significantly.
- New primary key values with referential integrity constraints are added. The primary key and foreign key tables should be re-analyzed.
- Table size notably changes relative to other tables it is joined to—for example, a table that was 50 times larger than another table is now only five times larger.
- A notable deviation in data distribution necessitates recalculating histograms—for example, an event causes abnormally high levels of trading for a particular stock.
- The database is inactive for an extended period of time.
Overhead Considerations

Running ANALYZE_STATISTICS is an efficient but potentially long-running operation. You can run it concurrently with queries and loads in a production environment. However, the function can incur considerable overhead on system resources (CPU and memory), at the expense of queries and load operations. To minimize overhead, run ANALYZE_STATISTICS on individual tables rather than the entire database, or on individual table columns.

Related Tools

You can diagnose and resolve many statistics-related issues by calling ANALYZE_WORKLOAD, which returns tuning recommendations.

If you update statistics and find that the query still performs poorly, run your query through the Database Designer and choose incremental as the design type. See Incremental Design.
Using Diagnostic Tools

Vertica provides several diagnostic tools. This section includes the following:

- Determining Your Version of Vertica
- Collecting Diagnostics: scrutinize Command
- Exporting a Catalog
- Exporting Profiling Data

Determining Your Version of Vertica

To determine which version of Vertica is installed on a host, log in to that host and type:

```
$ rpm -qa | grep vertica
```

The command returns the name of the installed package, which contains the version and build numbers. The following example indicates that both Vertica 7.0.x and Management Console 7.0.x are running on the targeted host:

```
[dbadmin@myhost01 ~]$ rpm -qa | grep vertica
vertica-7.0.0-0
vertica-console-7.0.0-0.x86_64
```

When you are logged in to your Vertica Analytic Database database, you can also run a query for the version only, by running the following command:

```
dbadmin=> SELECT version();
version
----------------------------------------------
Vertica Analytic Database v7.0.0-0
(1 row)
```

Collecting Diagnostics: scrutinize Command

The diagnostics tool scrutinize collects a broad range of information from a Vertica cluster. It also supports a range of options that let you control the amount and type of data that is collected. Collected data can include but is not limited to:
- Host diagnostics and configuration data
- Run-time state (number of nodes up or down)
- Log files from the installation process, the database, and the administration tools (such as, `vertica.log`, `dbLog`, `/opt/vertica/log/adminTools.log`)
- Error messages
- Database design
- System table information, such as system, resources, workload, and performance
- Catalog metadata, such as system configuration parameters
- Backup information

**Requirements**

`scrutinize` requires that a cluster be configured to support the Administration Tools utility. If Administration Tools cannot run on the initiating host, then `scrutinize` cannot run on that host.

**Running scrutinize**

You can run `scrutinize` as follows:

```
$ /opt/vertica/bin/scrutinize
```

Unqualified, `scrutinize` collects a wide range of information from all cluster nodes. It stores the results in a `.tar` file (`VerticaScrutinizeNumericID.tar`), with minimal effect on database performance. `scrutinize` output can help diagnose most issues and yet reduces upload size by omitting fine-grained profiling data.

**Command Options**

`scrutinize` options support the following tasks:
- Obtain version information about scrutinize and Vertica, and online help.
- Redirect output.
- Access a password-protected database.
- Control the scope of data collection.
- Upload results to Vertica Customer Support.

**Privileges**

You must have database administrator (dbadmin) privileges to run scrutinize. If you run scrutinize as root when the dbadmin user exists, Vertica returns an error.

**Disk Space Requirements**

scrutinize requires temporary disk space where it can collect data before posting the final compressed (.tar) output. How much space depends on variables such as the size of the Vertica log and extracted system tables, as well as user-specified options that limit the scope of information collected. Before scrutinize runs, it verifies that the temporary directory contains at least 1 GB of space; however, the actual amount needed can be much higher.

You can redirect scrutinize output to another directory. For details, see Redirecting scrutinize Output.

**Database Specification**

If multiple databases are defined on the cluster and more than one is active, or none is active, you must run scrutinize with one of the following options:

```
$ /opt/vertica/bin/scrutinize (--database=database | -d database)
```

If you omit this option when these conditions are true, scrutinize returns with an error.

**Database Identifier**

scrutinize generates a unique identifier for the database that it analyzes. This identifier is based on your Vertica license and the database creation time. scrutinize generates the same name for a given database each time it collects data on that database. It displays this name near the end of its standard output—in the following example, **PURPLE LEAD**:
Vertica Scrutinize Report
---------------------------------
Result Dir: /home/dbadmin/VerticaScrutinize.20140206144429

Gathered diagnostics for
Customer: Vertica Systems, Inc.
Database designation: PURPLE LEAD
Timestamp: 20140206144429

You can use this identifier when you interact with Vertica Customer Support. It is particularly useful in the following cases:

- A cluster hosts multiple databases.
- Different clusters host databases with the same name.

Informational Options

scrutinize supports two informational options that cannot be combined with any other options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--version</td>
<td>Obtains the version number of the Vertica server and the scrutinize version number, and then exits. For example:</td>
</tr>
<tr>
<td></td>
<td>=&gt; ! scrutinize --version</td>
</tr>
<tr>
<td></td>
<td>Scrutinize Version 9.0.1-20171119</td>
</tr>
<tr>
<td>--help</td>
<td>Lists all scrutinize options to the console, and then exits:</td>
</tr>
<tr>
<td>-h</td>
<td>=&gt; ! scrutinize -h</td>
</tr>
<tr>
<td></td>
<td>Usage: scrutinize [options]</td>
</tr>
<tr>
<td>Options:</td>
<td>show program's version number and exit</td>
</tr>
<tr>
<td>-version</td>
<td>show this help message and exit</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Skip tasks of a particular type. Provide a comma-separated lists of types to skip. Types are case-sensitive. Possible types are: Command, File, VerticalLog, DC, SystemTable, CatalogObject, Query, all.</td>
</tr>
<tr>
<td>-X LIST, --exclude-tasks=LIST</td>
<td></td>
</tr>
</tbody>
</table>

...
Redirecting scrutinize Output

By default, scrutinize uses the temporary directory /opt/vertica/tmp execution to compile output while it executes. On completing its collection, it saves the collection to a tar file to the current directory. You can redirect scrutinize output with two options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --tmpdir=path            | Directs temporary output to the specified path, where the following requirements apply to *path*:
|                         |   - The directory must have at least 1 GB of free space.                     |
|                         |   - You must have write permission to it.                                    |
| --output_dir=path -o path| Saves scrutinize results to a tar file in *path*. For example:               |
|                         | $ scrutinize --output_dir="/my_diagnostics/"                                |

scrutinize Security

scrutinize can specify user names and passwords as follows:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--user=username -U username</td>
<td>Specifies the dbadmin user name. By default, scrutinize uses the user name of the invoking user.</td>
</tr>
<tr>
<td>--password=password -P password</td>
<td>Sets the database password as an argument to the scrutinize command. Use this option if the administrator account (default dbadmin) has password authentication. If you omit this option on a password-protected database, scrutinize returns a warning, unless the environment variable VSQL_PASSWORD is set. Passwords with special characters must be enclosed with single quotes. For example: $ scrutinize -P '@password**' $ scrutinize --password='$password1*'</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-prompt-password</td>
<td>Specifies to prompt users for their database password before scrutinize begins to collect data.</td>
</tr>
</tbody>
</table>

## Data Collection Scope

`scrutinize` options let you control the scope of the data collection. You can specify the scope of the data collection according to the following criteria:

- **Time span**
- **Amount of data**, including its level of granularity
- **Specific nodes**
- **Types of data to include** and **exclude**

You can use these options singly or in combination, to achieve the desired level of granularity.

### Time Span

`scrutinize --begin` and `--end` options specify the time frame of diagnostic data that is collected from `vertica.log` and `editor.log`. You can use these options singly or together:

- **--begin**=`timestamp` Specifies to collect log data from `timestamp`.
- **--end**=`timestamp` Specifies to collect log data to `timestamp` or log end.

**Note:** These options cannot be combined with `--include_gzlogs`.

`timestamp` can be an offset from the current time, or an absolute time, as follows:

- **Absolute time:** `'YYYY-MM-DD HH:MM:SS'`
- **Offset time (from current timestamp):** `DdHhMmSs`

The `--begin` timestamp must precede the `--end` timestamp.

**Examples**

<table>
<thead>
<tr>
<th>Collect 24 hours of</th>
<th>$ scrutinize --begin='2017-11-19 00:00:00' --end='2017-11-20 00:00:00'</th>
</tr>
</thead>
</table>

Vertica Analytic Database (9.0.x)
data between 2017-11-19 and 2017-11-20:

<table>
<thead>
<tr>
<th>Scrutinize Version 9.0.1-20171119</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using default temporary directory</td>
</tr>
<tr>
<td>/tmp</td>
</tr>
<tr>
<td>Begin: 2017-11-19 00:00:00</td>
</tr>
<tr>
<td>End: 2017-11-20 00:00:00</td>
</tr>
</tbody>
</table>

Collect the last 2.5 hours of data:

<table>
<thead>
<tr>
<th>$ date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed Nov 22 10:38:05 EST 2017</td>
</tr>
<tr>
<td>$ scrutinize --begin=2h30m</td>
</tr>
<tr>
<td>Scrutinize Version 9.0.1-20171119</td>
</tr>
<tr>
<td>Using default temporary directory</td>
</tr>
<tr>
<td>/tmp</td>
</tr>
<tr>
<td>Begin: 2017-11-22 08:08:09</td>
</tr>
</tbody>
</table>

Collect data between the last 48 and 24 hours:

<table>
<thead>
<tr>
<th>$ date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed Nov 22 10:36:17 EST 2017</td>
</tr>
<tr>
<td>$ scrutinize --begin=2d --end=1d</td>
</tr>
<tr>
<td>Scrutinize Version 9.0.1-20171119</td>
</tr>
<tr>
<td>Using default temporary directory</td>
</tr>
<tr>
<td>/tmp</td>
</tr>
<tr>
<td>Begin: 2017-11-20 10:36:20</td>
</tr>
<tr>
<td>End: 2017-11-21 10:36:20</td>
</tr>
</tbody>
</table>

Collect data between 2017-11-19 and one day before the current time:

<table>
<thead>
<tr>
<th>$ date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed Nov 22 10:36:05 EST 2017</td>
</tr>
<tr>
<td>$ scrutinize --begin='2017-11-19 00:00:00' --end=1d</td>
</tr>
<tr>
<td>Scrutinize Version 9.0.1-20171119</td>
</tr>
<tr>
<td>Using default temporary directory</td>
</tr>
<tr>
<td>/tmp</td>
</tr>
<tr>
<td>Begin: 2017-11-19 00:00:00</td>
</tr>
<tr>
<td>End: 2017-11-21 10:36:16</td>
</tr>
</tbody>
</table>

Amount of Collected Data

Several options let you limit how much data scrutinize collects:

- **--by-second**
- **--by-minute**=`boolean-value`

Specifies the granularity of information that is collected from Data Collector tables with one of the following options:

- **--by-second**: Highest level of granularity, specifies to collect data down to the second.

- **--by-minute**=`boolean-value` where `boolean-value` is set to one of the following:
- Default setting, specifies to collect data down to the minute.
- Lowest level of granularity, specifies to collect data down to the hour.

For example, the following command collects the last five days of data down to the hour:

```bash
$ scrutinize --begin=5d --by-minute=no
```

This command collects the last hour's data down to the second:

```bash
$ scrutinize --begin=1h --by-second
```

**--get-files file-list**

Specifies extra files to collect, including globs, where `file-list` is a semicolon-delimited list of files.

**--include_gzlogs=num-files -z num-files**

Specifies to include `num-files` rotated log files (vertica.log*.gz) in the scrutinize output, where `num-files` can be one of the following:

- An integer specifies the number of rotated log files to collect.
- all specifies to collect all rotated log files.

By default, scrutinize includes three rotated log files.

For example the following command specifies to collect two rotated log files:

```bash
$ scrutinize --include_gzlogs=2
```

**--log-limit=limit -l limit**

Specifies how much data to collect from Vertica logs, where `limit` specifies, in gigabytes, how much log data to collect, starting from the most recent log entry. By default, scrutinize collects 1 GB of log data.

For example, the following command specifies to collect 4 GB of log data:
Node-Specific Collection

By default, scrutinize collects data from all cluster nodes. You can specify that scrutinize collect from individual nodes in two ways:

- `-local_diags -s` Specifies to collect diagnostics only from the host on which scrutinize was invoked.

  Tip: To collect data from multiple nodes in the cluster, use the --hosts option.

- `--hosts=host-list -n host-list` Specifies to collect diagnostics only from the hosts specified in `host-list`, where `host-list` is a comma-separated list of IP addresses or host names.

  For example:

  ```
  $ scrutinize --hosts=127.0.0.1,host_3,host_1
  ```

Types of Data to Include

scrutinize provides several options that let you specify the type of data to collect:

- `--debug` Collects debug information for the log.

- `--diag-dump` Limits the collection to database design, system tables, and Data Collector tables. Use this option to collect data to analyze system performance.

- `--diagnostics` Limits the collection to log file data and output from commands that are run against Vertica and its host system. Use this option to collect data to evaluate unexpected behavior in your Vertica system.

- `--include-ros-info` Includes ROS related information from system tables.

- `--no-active-queries` Specifies to exclude diagnostic information from system tables and Data Collector tables about currently running
queries. By default, scrutinize collects this information (--with-active-queries).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--tasks=<strong>tasks</strong></td>
<td>Specifies that scrutinize gather diagnostics on one or more tasks, as specified in a file or JSON list. This option is typically used together with --exclude.</td>
</tr>
<tr>
<td>-T tasks</td>
<td>Note: Use this option only in consultation with Vertica Customer Support.</td>
</tr>
<tr>
<td>--type=<strong>type</strong></td>
<td>Specifies the type of diagnostics collection to perform, where <strong>type</strong> can be one of the following arguments:</td>
</tr>
<tr>
<td>-t type</td>
<td>- profiling: Gather profiling data.</td>
</tr>
<tr>
<td></td>
<td>- context: Gather summary information.</td>
</tr>
<tr>
<td>--with-active-queries</td>
<td>The default setting, specifies to include diagnostic information from system tables and Data Collector tables about currently running queries. To omit this data, use --no-active-queries.</td>
</tr>
</tbody>
</table>

### Types of Data to Exclude

**scrutinize** options also let you specify the types of data to exclude from its collection:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--exclude=<strong>tasks</strong></td>
<td>Excludes one or more types of tasks from the diagnostics collection, where <strong>tasks</strong> is a comma-separated list of the tasks to exclude.</td>
</tr>
<tr>
<td>-X tasks</td>
<td>Note: This option is typically used only in consultation with your Vertica Customer Support contact.</td>
</tr>
</tbody>
</table>

Specify the tasks to exclude with the following case-insensitive arguments:

- all: All default tasks
- DC: Data Collector tables
- File: Log files from the installation process, the database, and Administration Tools, such as vertica.log, dbLog, and adminTools.log
- **VerticaLog**: Vertica logs
- **CatalogObject**: Vertica catalog metadata, such as system configuration parameters
- **SystemTable**: Vertica system tables that contain information about system, resources, workload, and performance
- **Query**: Vertica metafunctions that use vsql to connect to the database, such as `EXPORT_CATALOG()`
- **Command**: Operating system information, such as the length of time that a node has been up

---

<table>
<thead>
<tr>
<th><code>-no-active-queries</code></th>
<th>Specifies to omit diagnostic information from system tables and Data Collector tables about currently running queries. By default, <code>scrutinize</code> always collects active query information (<code>-with-active-queries</code>).</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-vsq1-off -v</code></td>
<td>Excludes Query and SystemTable tasks, which are used to connect to the database. This option can help you deal with problems that occur during an upgrade, and is typically used in the following cases:</td>
</tr>
<tr>
<td></td>
<td>- Vertica is running but is slow to respond.</td>
</tr>
<tr>
<td></td>
<td>- You haven't yet created a database but need help troubleshooting other cluster issues.</td>
</tr>
</tbody>
</table>

---

**Uploading scrutinize Results**

`scrutinize` provides several options for uploading data to Vertica customer support.
Upload Packaging

When you use an upload option, scrutinize does not bundle all output in a single tar file. Instead, each node posts its output directly to the specified URL as follows:

1. Uploads a smaller, context file, enabling Customer Support to review high-level information.

2. On completion of scrutinize execution, uploads the complete diagnostics collection.

Upload Prerequisites

Before you run scrutinize with an upload option:

- Install the cURL program installed in the path for the database administrator user who is running scrutinize.

- Verify each node in the cluster can make an HTTP or FTP connection directly to the Internet.

Upload Options

Note: Two options upload scrutinize output to a Vertica support-provided URL or FTP address: --auth-upload and --url. Each option authenticates the upload differently, as noted below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--auth-upload=URL</td>
<td>Uses your Vertica license to authenticate with the Vertica server, by uploading your customer name. Customer Support uses this information to verify your identity on receiving your uploaded file. This option requires a valid Vertica Premium Edition license.</td>
</tr>
<tr>
<td>-A URL</td>
<td>For example: $ scrutinize -U username -P 'password' --auth-upload=&quot;URL&quot;</td>
</tr>
<tr>
<td>--url=URL -u URL</td>
<td>Requires URL to include a user name and password that is supplied by Vertica Customer Support.</td>
</tr>
<tr>
<td></td>
<td>For example: $ scrutinize -U username -P 'password' --</td>
</tr>
</tbody>
</table>
include a message with the scrutinize output, where message is one of the following:

- "message text"
  A message string. For example:
  
  $ scrutinize --message="re: case number #ABC-12345"

- "file-path"
  A path to a text file. For example:
  
  $ scrutinize --message="/path/to/msg.txt"

- PROMPT
  Opens an input stream. scrutinize reads input until you type a period (.) on a new line. This closes the input stream, and scrutinize writes the message to the collected output.

  $ scrutinize --message=PROMPT
  Enter reason for collecting diagnostics; end with '.' on a line by itself:
  Query performance degradation noticed around 9AM EST on Saturday.
  Vertica Scrutinize Report
  ---------------------------------------------------------------
  Result Dir: /home/dbadmin/VerticaScrutinize.20131126083311
  ...

The message is set in the output directory, in reason.txt. If no message is specified, scrutinize generates the default message Unknown reason for collection. Messages typically include the following information:

- Reason for gathering/submitting diagnostics.

- Support-supplied case number and other issue-specific information, to help Vertica Customer Support identify your case and analyze the problem.
Troubleshooting scrutinize

The troubleshooting advice in this section can help you resolve common issues that you might encounter when using scrutinize.

Collection Time Is Too Slow

To speed up collection time, omit system tables when running an instance of scrutinize. Be aware that collecting from fewer nodes does not necessarily speed up the collection process.

Output Size Is Too Large

Output size depends on system table size and vertica log size.

To create a smaller scrutinize output, omit some system tables or truncate the vertica log. For more information, see Narrowing the Scope of scrutinize Data Collection.

System Tables Not Collected on Databases with Password

Running scrutinize on a password-protected database might require you to supply a user name and password:

```
$ scrutinize -U username -P 'password'
```

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the EXPORT_CATALOG function in the SQL Reference Manual for details.
Exporting Profiling Data

The diagnostics audit script gathers system table contents, design, and planning objects from a running database and exports the data into a file named "./diag_dump_<timestamp>.tar.gz", where <timestamp> denotes when you ran the script.

If you run the script without parameters, you will be prompted for a database password.

Syntax

/opt/vertica/scripts/collect_diag_dump.sh [ command... ]

Parameters

<table>
<thead>
<tr>
<th>command</th>
<th>-U</th>
<th>User name, typically the database administrator account, dbadmin.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-w</td>
<td>Database password.</td>
</tr>
<tr>
<td></td>
<td>-c</td>
<td>Includes a compression analysis, resulting in a longer script execution time.</td>
</tr>
</tbody>
</table>

Example

The following command runs the audit script with all arguments:

```
$ /opt/vertica/scripts/collect_diag_dump.sh -U dbadmin -w password -c
```
Profiling Database Performance

You can profile database performance to evaluate how efficiently queries execute. For example, profiles can provide the following information:

- How much memory and how many threads each operator is allocated.
- How data flows through each operator at different points in time during query execution.
- Whether a query is network bound.

Profiling information can help you evaluate query performance and determine whether to rewrite the query. You can use profiling when considering projection design issues, such as segmentation and sort order.

The topics in this section focus on obtaining profile data via vsq1 statements. The Vertica Management Console also provides an easy-to-read view of query profile data.

Profiling Categories

Vertica divides profiling data into three categories and captures this information in several system tables. You can query those tables for specific performance information, as shown in the following table:

<table>
<thead>
<tr>
<th>Profiling Category</th>
<th>System Tables</th>
<th>Description of Profiled Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>SESSION_PROFILES</td>
<td>General information about query execution on each node during the current session. For example, you can find out how many statements ran successfully and unsuccessfully, how many locks were granted and deadlocks encountered, and so on.</td>
</tr>
<tr>
<td>Queries</td>
<td>QUERY_PLAN_PROFILES</td>
<td>Query-specific information, such as query string and duration of execution, divided between two system tables:</td>
</tr>
<tr>
<td></td>
<td>QUERY_PROFILES</td>
<td>- QUERY_PLAN_PROFILES: Real-time status for each query plan path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- QUERY_PROFILES: Query information.</td>
</tr>
<tr>
<td>Profiling Category</td>
<td>System Tables</td>
<td>Description of Profiled Data</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Execution engine</td>
<td>EXECUTION_ENGINE_PROFILES</td>
<td>Information on execution engine performance.</td>
</tr>
</tbody>
</table>

Note: If execution engine profiling is disabled, Vertica saves no data in EXECUTION_ENGINE_PROFILES. However, if query and session profiling is disabled, Vertica saves some data in QUERY_PROFILES, QUERY_PLAN_PROFILES, and SESSION_PROFILES.

For each category, you can enable profiling globally, for the entire database, or only for the current session. For more information, see Enabling and Disabling Profiling.

## Enabling and Disabling Profiling

You can enable and disable profiling for a specific category globally (for the entire database) and the current session. Use SHOW_PROFILING_CONFIG to determine whether profiling is enabled. Output from this command shows:

- At what scope (global and session) profiling is enabled.
- For what categories profiling is enabled.

Note: If profiling is enabled globally for a given category, enabling or disabling session profiling has no effect on whether Vertica gathers profile data for that category. The session profiling setting enables profiling for a given category only when global profiling is disabled for the same category.

In the following example, SHOW_PROFILING_CONFIG shows that profiling is enabled globally (Global on) across all profiling categories—session, EE (execution engine), and query. For the current session, it is disabled (Session off) across all categories:

```
=> SELECT SHOW_PROFILING_CONFIG();
SHOW_PROFILING_CONFIG
----------------------------------------
Session Profiling: Session off, Global on
EE Profiling: Session off, Global on
Query Profiling: Session off, Global on
(1 row)
```
Enable or Disable Global Profiling

\[
\text{ALTER DATABASE } db\text{-name SET profiling-category = \{0 | 1\}}
\]

where \textit{profiling-category} is set to one of the following arguments:

<table>
<thead>
<tr>
<th>Use this argument...</th>
<th>To specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalSessionProfiling</td>
<td>Session profile data</td>
</tr>
<tr>
<td>GlobalQueryProfiling</td>
<td>Query data</td>
</tr>
<tr>
<td>GlobalEEProfiling</td>
<td>Execution engine data</td>
</tr>
</tbody>
</table>

For example, the following statement globally enables query profiling:

\[
=> \text{ALTER DATABASE mydb SET GlobalQueryProfiling = 1;}
\]

Enable or Disable Session Profiling

\[
\text{ENABLE_PROFILING( profiling-category )}
\]
\[
\text{DISABLE_PROFILING( profiling-category )}
\]

where \textit{profiling-category} is set to one of the following arguments:

<table>
<thead>
<tr>
<th>Use this argument...</th>
<th>To specify...</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Session profile data</td>
</tr>
<tr>
<td>query</td>
<td>Query data</td>
</tr>
<tr>
<td>ee</td>
<td>Execution engine data</td>
</tr>
</tbody>
</table>

For example, the following statement enables session-scoped profiling for the execution run of each query:

\[
=> \text{SELECT ENABLE_PROFILING('ee');}
\]
\[
\text{ENABLE_PROFILING}
\]
\[
\text{-----------------------------}
\]
\[
\text{EE Profiling Enabled}
\]
\[
(1 \text{ row})
\]

Vertica Analytic Database (9.0.x)
What to look for in query profiles

Profile data can show data skew, when some nodes are processing more data than others. The rows produced counter in the system table EXECUTIONENGINEProfiles shows how many rows have been processed by each operator. Comparing the rows produced across all nodes for a given operator reveals if there is a data skew issue.

Note: Some profiling operators in EXECUTIONENGINEProfiles are generic, such as JOIN. The EXPLAIN-generated query plan includes more details about the specific join that is executing.

Real-Time Profiling

With real-time profiling, you can monitor long-running queries while they execute.

Real-time profiling counters are available for all statements while they execute, including internal operations such as mergeout, recovery, and refresh. Unless you explicitly enable profiling using the keyword PROFILE on a specific SQL statement, or generally enable profiling for the database and/or the current session, profiling counters are unavailable after the statement completes.

Queries for real-time profiling data require a transaction ID. If the transaction executes multiple statements, the query also requires a statement ID to identify the desired statement. You obtain transaction and statement IDs the SYSTEMSESSIONS system table from columns transaction_id and statement_id, respectively:

```
=> SELECT transaction_id, statement_id FROM SYSTEM_SESSIONS;
```

Profiling Counters

The EXECUTIONENGINEProfiles system table contains available profiling counters for internal operations and user statements.

Useful counters include:

- Execution time (µs)
- Rows produced
- Total merge phases
- Completed merge phases
- Current size of temp files (bytes)

You can view all available counters by querying `EXECUTION_ENGINE_PROFILES`:

```
=> SELECT DISTINCT(counter_name) FROM EXECUTION_ENGINE_PROFILES;
```

To monitor the profiling counters, you can run a command like the following using a retrieved transaction ID (a00000000027):

```
=> SELECT * FROM execution_engine_profiles
    WHERE TO_HEX(transaction_id)='a00000000027'
    AND counter_name = 'execution time (us)'
    ORDER BY node_name, counter_value DESC;
```

The following example finds operators with the largest execution time on each node:

```
=> SELECT node_name, operator_name, counter_value execution_time_us
    FROM V_MONITOR.EXECUTION_ENGINE_PROFILES
    WHERE counter_name='execution time (us)'
    LIMIT 1 OVER(PARTITION BY node_name ORDER BY counter_value DESC);
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>operator_name</th>
<th>execution_time_us</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node001</td>
<td>Join</td>
<td>131906</td>
</tr>
<tr>
<td>v_vmart_node002</td>
<td>Join</td>
<td>227778</td>
</tr>
<tr>
<td>v_vmart_node003</td>
<td>NetworkSend</td>
<td>524080</td>
</tr>
</tbody>
</table>
(3 rows)

**Query Plan Profiles**

You can obtain query-specific data by profiling the query statement and evaluating the data that is captured in system tables `QUERY_PLAN_PROFILES` and `EXECUTION_ENGINE_PROFILES`. For example, you can query `QUERY_PLAN_PROFILES` to determine how much time a query spends on each query plan operation. For details, see Profiling Query Plans.

**Linux watch Command**

You can use the Linux `watch` command to monitor long-running queries at frequent intervals. Common use cases include:
• Observing executing operators within a query plan on each Vertica cluster node.

• Monitoring workloads that might be unbalanced among cluster nodes—for example, some nodes become idle while others are active. Such imbalances might be caused by data skews or by hardware issues.

In the following example, `watch` queries operators with the largest execution time on each node. The command specifies to re-execute the query each second:

```
watch -n 1 -d "vsql VMart -c"SELECT node_name, operator_name, counter_value execution_time_us
FROM v_monitor.execution_engine_profiles WHERE counter_name='execution time (us)'
LIMIT 1 OVER(PARTITION BY node_name ORDER BY counter_value DESC);
```

Every 1.0s: vsql VMart -c"SELECT node_name, operator_name, counter_value execution_time_us FROM v_monitor.execution_engine_profiles WHERE counter_name='execution time (us)'
LIMIT 1 OVER(PARTITION BY node_name ORDER BY counter_value DESC);

<table>
<thead>
<tr>
<th>node_name</th>
<th>operator_name</th>
<th>execution_time_us</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>Root</td>
<td>110266</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>UnionAll</td>
<td>38932</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>Scan</td>
<td>22058</td>
</tr>
</tbody>
</table>

(3 rows)

### Profiling Single Statements

To profile a single statement, prefix it with `PROFILE`. You can profile a query (SELECT) statement, or any DML statement such as `INSERT`, `UPDATE`, `COPY`, and `MERGE`. The statement returns with a profile summary:

• Profile identifiers `transaction_id` and `statement_id`

• Initiator memory for the query

• Total memory required

For example:

```
=> PROFILE SELECT customer_name, annual_income FROM public.customer_dimension
    WHERE (customer_gender, annual_income) IN (SELECT customer_gender, MAX(annual_income)
    FROM public.customer_dimension GROUP BY customer_gender);
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996278193955 and
statement_id=1;
NOTICE 3557: Initiator memory for query: [on pool pool1: 996147 KB, minimum: 746280 KB]
NOTICE 5077: Total memory required by query: [996147 KB]
customer_name | annual_income
---------------|-----------------
You can use the profile identifiers `transaction_id` and `statement_id` to obtain detailed profile information for this query from system tables `EXECUTION_ENGINE_PROFILES` and `QUERY_PLAN_PROFILES`. For example:

```
=> SELECT path_id, path_line::VARCHAR(68), running_time FROM v_monitor.query_plan_profiles
    WHERE transaction_id=45035996278193955 AND statement_id=1 ORDER BY path_id, path_line_index;
```

<table>
<thead>
<tr>
<th>path_id</th>
<th>path_line</th>
<th>running_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>++JOIN HASH [Semi] [Cost: 644, Rows: 25K] (PATH ID: 1)</td>
<td>00:00:00.009077</td>
</tr>
<tr>
<td>1</td>
<td>Join Cond: (customer_dimension.customer_gender = VAL(2)) AND (customer_dimension.customer_name)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Materialize at Output: customer_dimension.customer_name</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Execute on: Query Initiator</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>--- Outer -&gt; STORAGE ACCESS for customer_dimension [Cost: 149, Row</td>
<td>00:00:00.008763</td>
</tr>
<tr>
<td>2</td>
<td>Projection: public.customer_dimension_DBD_1_rep_VMartDesign</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Materialize: customer_dimension.customer_gender, customer_d</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Execute on: Query Initiator</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Runtime Filters: (SIP1(HashJoin): customer_dimension.customer_gender)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>++GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 159, Rows: 3]</td>
<td>00:00:00.006566</td>
</tr>
<tr>
<td>4</td>
<td>Aggregates: max(customer_dimension.annual_income)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Group By: customer_dimension.customer_gender</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Execute on: Query Initiator</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>++ STORAGE ACCESS for customer_dimension [Cost: 149, Rows:</td>
<td>00:00:00.006943</td>
</tr>
<tr>
<td>5</td>
<td>Projection: public.customer_dimension_DBD_1_rep_VMartDesign</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Materialize: customer_dimension.customer_gender, customer_d</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Execute on: Query Initiator</td>
<td></td>
</tr>
</tbody>
</table>

(17 rows)

**Sample Views for Counter Information**

The `EXECUTIONENGINE_PROFILES` table contains the data for each profiling counter as a row within the table. For example, the execution time (us) counter is in one row, and the rows produced counter is in a second row. Since there are many different profiling counters, many rows of profiling data exist for each operator. Some sample views are installed by default to simplify the viewing of profiling counters.

**Running scripts to create the sample views**

The following script creates the `v_demo` schema and places the views in that schema.

```
/opt/vertica/scripts/demo_eeprof_view.sql
```
Viewing counter values using the sample views

There is one view for each of the profiling counters to simplify viewing of a single counter value. For example, to view the execution time for all operators, issue the following command from the database:

```sql
=> SELECT * FROM v_demo.eeprof_execution_time_us;
```

To view all counter values available for all profiled queries:

```sql
=> SELECT * FROM v_demo.eeprof_counters;
```

To select all distinct operators available for all profiled queries:

```sql
=> SELECT * FROM v_demo.eeprof_operators;
```

Combining sample views

These views can be combined:

```sql
=> SELECT * FROM v_demo.eeprof_execution_time_us
  NATURAL LEFT OUTER JOIN v_demo.eeprof_rows_produced;
```

To view the execution time and rows produced for a specific transaction and statement_id ranked by execution time on each node:

```sql
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank
  WHERE transaction_id=45035996273709699
  AND statement_id=1
  ORDER BY transaction_id, statement_id, node_name, rk;
```

To view the top five operators by execution time on each node:

```sql
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank
  WHERE transaction_id=45035996273709699
  AND statement_id=1 AND rk<=5
  ORDER BY transaction_id, statement_id, node_name, rk;
```
Profiling Query Plans

To monitor real-time flow of data through a query plan, query system tables EXECUTION_ENGINE_PROFILES and QUERY_PLAN_PROFILES. These tables provides data on how Vertica executed a query plan and its individual paths:

- EXECUTION_ENGINE_PROFILES summarizes query execution runs.
- QUERY_PLAN_PROFILES shows the real-time flow of data, and the time and resources consumed for each query plan path.

Each query plan path has a unique ID, as shown in the following EXPLAIN output fragment.

```
+--JOIN MERGEJOIN(inputs presorted) [Cost: 7, Rows: 1 (NO STATISTICS)] (PATH ID: 1) INNER (BROADCAST)
  | Join Cond: (t1.x = t2.y)
  | Materialize at output: t2.z
  | Execute on: All Nodes
  +-- Outer -> STORAGE ACCESS for t2 [Cost: 3, Rows: 3 (NO STATISTICS)] (PATH ID: 2)
  ...
```

Both tables provide path-specific data. For example, QUERY_PLAN_PROFILES provides high-level data for each path, which includes:

- Length of a query operation execution
- How much memory that path's operation consumed
- Size of data sent/received over the network

For example, you might observe that a GROUP BY HASH operation executed in 0.2 seconds using 100MB of memory.

Requirements

Real-time profiling minimally requires the ID of the transaction to monitor. If the transaction includes multiple statements, you also need the statement ID. You can get statement and transaction IDs by issuing PROFILE on the query to profile. You can then use these identifiers to query system tables EXECUTION_ENGINE_PROFILES and QUERY_PLAN_PROFILES.

For more information, see Profiling Single Statements.
Getting Query Plan Status for Small Queries

Real-time profiling counters, stored in the EXECUTION_ENGINE_PROFILES system table, are available for all currently executing statements—including internal operations, such as a mergeout.

Profiling counters are available after query execution has completed if any of the following conditions are true:

- The query was run via the PROFILE <query> command
- Systemwide profiling has been enabled through the ENABLE_PROFILING() function
- The query took longer than two seconds to run

Profiling counters are saved in the EXECUTION_ENGINE_PROFILES system table until the storage quota has been exceeded.

Here's an example:

1. Profile the query to get the transaction_id and statement_id from EXECUTION_ENGINE_PROFILES; for example:

```sql
=> PROFILE SELECT * FROM t1 JOIN t2 ON t1.x = t2.y;
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996273955065 and statement_id=4;
NOTICE 3557: Initiator memory for query: [on pool general: 248544 KB, minimum: 248544 KB]
NOTICE 5077: Total memory required by query: [248544 KB]
 x | y | z
---+-------
 3 | 3 | three
(1 row)
```

2. Query the QUERY_PLAN_PROFILES system table.

Note: For best results, sort on the transaction_id, statement_id, path_id, and path_line_index columns.

```sql
=> SELECT ... FROM query_plan_profiles
   WHERE transaction_id=45035996273955065 and statement_id=4;
ORDER BY transaction_id, statement_id, path_id, path_line_index;
```
Getting Query Plan Status for Large Queries

Real-time profiling is designed to monitor large (long-running) queries. Take the following steps to monitor plans for large queries:

1. Get the statement and transaction IDs for the query plan you want to profile by querying the `CURRENT_SESSION` system table:

```
=> SELECT transaction_id, statement_id from current_session;
transaction_id | statement_id
------------- | -----------
45035996273955001 | 4
(1 row)
```

2. Run the query:

```
=> SELECT * FROM t1 JOIN t2 ON x=y JOIN ext on y=z;
```

3. Query the `QUERY_PLAN_PROFILES` system table, and sort on the `transaction_id`, `statement_id`, `path_id`, and `path_line_index` columns.

```
=> SELECT ... FROM query_plan_profiles WHERE transaction_id=45035996273955001 and statement_id=4 ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

You can also use the Linux `watch` command to monitor long-running queries (see Real-Time Profiling).

Example

The following series of commands creates a table for a long-running query and then runs the `QUERY_PLAN_PROFILES` system table:

```
=> CREATE TABLE longq(x int);
CREATE TABLE
=> COPY longq FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> 5
>> 6
>> 7
```
=> INSERT INTO longq SELECT f1.x+f2.x+f3.x+f4.x+f5.x+f6.x+f7.x
   FROM longq f1
   CROSS JOIN longq f2
   CROSS JOIN longq f3
   CROSS JOIN longq f4
   CROSS JOIN longq f5
   CROSS JOIN longq f6
   CROSS JOIN longq f7;

OUTPUT
-----------
10000000
(1 row)
=> COMMIT;
COMMIT

Suppress query output on the terminal window by using the vsqI \o command:

=> \o /home/dbadmin/longQprof

Query the new table:

=> SELECT * FROM longq;

Get the transaction and statement IDs:

=> SELECT transaction_id, statement_id from current_session;
   transaction_id | statement_id
   ---------------|---------------
   45035996273955021 | 4
(1 row)

Turn off the \o command so that Vertica continues to save query plan information to the file you specified. Alternatively, leave it on and examine the file after you query the QUERY_PLAN_PROFILES system table.

=> \o

Query the QUERY_PLAN_PROFILES system table:

=> SELECT
   transaction_id,
   statement_id,
   path_id,
   path_line_index,
   is_executing,
   running_time,
   path_line
FROM query_plan_profiles
WHERE transaction_id=45035996273955021 AND statement_id=4
Improving Readability of QUERY_PLAN_PROFILES Output

Output from the QUERY_PLAN_PROFILES table can be very wide because of the path_line column. To facilitate readability, query QUERY_PLAN_PROFILES using one or more of the following options:

- **Sort output by** transaction_id, statement_id, path_id, and path_line_index:

  ```sql
  => SELECT ... FROM query_plan_profiles
      WHERE ...
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
  ```

- **Use column aliases to decrease column width:**

  ```sql
  => SELECT statement_id AS sid, path_id AS id, path_line_index AS order,
      is_started AS start, is_completed AS end, is_executing AS exe,
      running_time AS run, memory_allocated_bytes AS mem,
      read_from_disk_bytes AS read, received_bytes AS rec,
      sent_bytes AS sent, FROM query_plan_profiles
      WHERE transaction_id=45035996273910558 AND statement_id=3
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
  ```

- **Use the vsql \o command to redirect EXPLAIN output to a file:**

  ```sql
  => \o /home/dbadmin/long-queries
  => EXPLAIN SELECT * FROM customer_dimension;
  => \o
  ```

Managing Query Profile Data

Vertica retains data for queries until the storage quota for the table is exceeded, when it automatically purges the oldest queries to make room for new ones. You can clear profiled data by manually calling one of the following functions:

- **CLEAR_PROFILING()** clears profiled data from memory. For example, the following command clears profiling for general query-run information, such as the query strings used and the duration of queries.
**Configuring data retention policies**

Vertica retains the historical data it gathers based on retention policies, which a superuser can configure. See Retaining Monitoring Information.

**Analyzing Suboptimal Query Plans**

If profiling uncovers a suboptimal query, invoking one of the following functions might help:

- **ANALYZE_WORKLOAD** analyzes system information held in system tables and provides tuning recommendations that are based on a combination of statistics, system and data collector events, and database-table-projection design.

- **ANALYZE_STATISTICS** collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table or column.

You can also run your query through the Database Designer. See Incremental Design.

**Labeling Queries**

To quickly identify queries for profiling and debugging purposes, include the LABEL hint.

LABEL hints are valid in the following statements:

- **DELETE**

- **INSERT**

- **MERGE**
- SELECT
- UPDATE

- UNION: Valid in the UNION's first SELECT statement. Vertica ignores labels in subsequent SELECT statements.

For example:

```sql
SELECT /*+label(myselectquery)*/ COUNT(*) FROM t;
INSERT /*+label(myinsertquery)*/ INTO t VALUES(1);
```

After you add a label to one or more queries, query the `QUERY_PROFILES` system table to see which queries ran with your supplied labels. The `QUERY_PROFILES` system table IDENTIFIER column returns the user-defined label that you previously assigned to a query. You can also obtain other query-specific data that can be useful for querying other system tables, such as transaction IDs.

For example:

```sql
=> SELECT identifier, query FROM query_profiles;

<table>
<thead>
<tr>
<th>identifier</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>myselectquery</td>
<td>SELECT /<em>+label(myselectquery)</em>/ COUNT(*) FROM t;</td>
</tr>
<tr>
<td>myinsertquery</td>
<td>INSERT /<em>+label(myinsertquery)</em>/ INTO t VALUES(1);</td>
</tr>
<tr>
<td>myupdatequery</td>
<td>UPDATE /<em>+label(myupdatequery)</em>/ t SET a = 2 WHERE a = 1;</td>
</tr>
<tr>
<td>mydeletequery</td>
<td>DELETE /<em>+label(mydeletequery)</em>/ FROM t WHERE a = 1;</td>
</tr>
<tr>
<td></td>
<td>SELECT identifier, query from query_profiles;</td>
</tr>
</tbody>
</table>
```

(5 rows)
About Locale

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate GROUP BY and ORDER BY clauses, joins, and the analytic ORDER BY clause.

The default locale for a Vertica database is en_US@collation=binary (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default en_US@collation=binary collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to en_US_POSIX. You can set the locale back to the default locale and collation by issuing the vsql meta-command \locale. For example:

```plaintext
=> set locale to '';
INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation-binary;
```

You can set locale through ODBC, JDBC, and ADO.net.

Vertica locale specifications follow a subset of the Unicode LDML standard as implemented by the ICU library.

Locale Handling in Vertica

The following sections describes how Vertica handles locale.

Session Locale

Locale is session-scoped and applies only to queries executed in that session. You cannot specify locale for individual queries. When you start a session it obtains its locale from the
configuration parameter `DefaultSessionLocale`.

**Query Restrictions**

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- When one or more of the left-side `NOT IN` columns is `CHAR` or `VARCHAR`, multi-column `NOT IN` subqueries are not supported. For example:

```sql
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
   ERROR: Multi-expression NOT IN subquery is not supported because a left hand expression could be NULL
```

**Note:** Even if columns `test.x` and `test.y` have a `NOT NULL` constraint, an error occurs.

- If the outer query contains a `GROUP BY` clause on a `CHAR` or `VARCHAR` column, correlated `HAVING` clause subqueries are not supported. In the following example, the `GROUP BY` `x` in the outer query causes the error:

```sql
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
    IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a');
   ERROR: subquery uses ungrouped column "t.x" from outer query
```

- Subqueries that use analytic functions in the `HAVING` clause are not supported. For example:

```sql
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1) FROM test
    GROUP BY x HAVING x IN (SELECT MAX(x) FROM test);
   ERROR: Analytics query with having clause expression that involves aggregates and subquery is not supported
```

**Collation and Projections**

Projection data is sorted according to the default `en_US@collation=binary` collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by `col1` according to the binary collation:
CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY col1;

In such cases, straße and strasse are not stored near each other on disk.

Sorting by binary collation also means that sort optimizations do not work in locales other than binary. Vertica returns the following warning if you create tables or projections in a non-binary locale:

WARNING: Projections are always created and persisted in the default Vertica locale. The current locale is de_DE

Non-Binary Locale Input Handling

When the locale is non-binary, Vertica uses the COLLATION function to transform input to a binary string that sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

\[ \text{result
column\nwidth} = \text{input\noctet\nwidth} \times \text{CollationExpansion} + 4 \]

The default value of configuration parameter CollationExpansion is 5.

Character Data Type Handling

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-binary locale.

- The maximum length parameter for VARCHAR and CHAR data type refers to the number of octets (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 bytes per character, depending on the data.

Specifying Locale: Long Form

Vertica supports long forms that specify the collation keyword. Vertica extends long-form processing to accept collation arguments.
Syntax

[Language][_script][_country][_variant][@collation-spec]

Note: The following syntax options apply:

- Locale specification strings are case insensitive. For example, `en_us` and `EN_US`, are equivalent.
- You can substitute underscores with hyphens. For example: `[-script]`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>A two- or three-letter lowercase code for a particular language. For example, Spanish is <code>es</code> English is <code>en</code> and French is <code>fr</code>. The two-letter language code uses the ISO-639 standard.</td>
</tr>
<tr>
<td>_script</td>
<td>An optional four-letter script code that follows the language code. If specified, it should be a valid script code as listed on the Unicode ISO 15924 Registry.</td>
</tr>
<tr>
<td>_country</td>
<td>A specific language convention within a generic language for a specific country or region. For example, French is spoken in many countries, but the currencies are different in each country. To allow for these differences among specific geographical, political, or cultural regions, locales are specified by two-letter, uppercase codes. For example, FR represents France and CA represents Canada. The two letter country code uses the ISO-3166 standard.</td>
</tr>
<tr>
<td>_variant</td>
<td>Differences may also appear in language conventions used within the same country. For example, the Euro currency is used in several European countries while the individual country's currency is still in circulation. To handle variations inside a language and country pair, add a third code, the variant code. The variant code is arbitrary and completely application-specific. ICU adds _EURO to its locale designations for locales that support the Euro currency. Variants can have any number of underscored key words. For example, <code>EURO_WIN</code> is a variant for the Euro currency on a Windows computer.</td>
</tr>
</tbody>
</table>
Another use of the variant code is to designate the Collation (sorting order) of a locale. For instance, the es__TRADITIONAL locale uses the traditional sorting order which is different from the default modern sorting of Spanish.

| @collation-spec | Vertica only supports the keyword collation, as follows:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>@collation=collation-type[:arg]...</td>
<td>Collation can specify one or more semicolon-delimited arguments, described below.</td>
</tr>
<tr>
<td></td>
<td><strong>collation-type</strong> is set to one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• big5han: Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese).</td>
</tr>
<tr>
<td></td>
<td>• dict: For a dictionary-style ordering (such as in Sinhala).</td>
</tr>
<tr>
<td></td>
<td>• direct: Hindi variant.</td>
</tr>
<tr>
<td></td>
<td>• gb2312/gb2312han: Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese).</td>
</tr>
<tr>
<td></td>
<td>• phonebook: For a phonebook-style ordering (such as in German).</td>
</tr>
<tr>
<td></td>
<td>• pinyin: Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese).</td>
</tr>
<tr>
<td></td>
<td>• reformed: Reformed collation (such as in Swedish).</td>
</tr>
<tr>
<td></td>
<td>• standard: The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales.</td>
</tr>
<tr>
<td></td>
<td>• stroke: Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported.</td>
</tr>
<tr>
<td></td>
<td>• traditional: For a traditional-style ordering (such as in Spanish).</td>
</tr>
<tr>
<td></td>
<td>• unihan: Pinyin ordering for Latin, Unihan radical-stroke ordering</td>
</tr>
</tbody>
</table>
for CJK characters (used in Chinese) not supported.

- binary: Vertica default, providing UTF-8 octet ordering.

Notes:

- Collations might default to root, the ICU default collation.

- Invalid values of the collation keyword and its synonyms do not cause an error. For example, the following does not generate an error. It simply ignores the invalid value:

  => \locale en_GB@collation=xyz
INFO 2567: Canonical locale: 'en_GB@collation=xyz'
Standard collation: 'LEN'
English (United Kingdom, collation=xyz)

For more about collation options, see Unicode Locale Data Markup Language (LDML).

Collation Arguments

collation can specify one or more of the following arguments:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short form</th>
<th>Description</th>
</tr>
</thead>
</table>
| colstrength   | S          | Sets the default strength for comparison. This feature is locale dependent. Set colstrength to one of the following:  
- 1 | primary: Ignores case and accents. Only primary differences are used during comparison—for example, a versus z.  
- 2 | secondary: Ignores case. Only secondary and above differences are considered for comparison—for example, different accented forms of the same base letter such as a versus  
\u00E4.  
- 3 | tertiary (default): Only tertiary differences and higher are considered for |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>comparison. Tertiary comparisons are typically used to evaluate case differences—for example, Z versus z.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 4</td>
</tr>
<tr>
<td>colAlternate</td>
<td>A</td>
<td>Sets alternate handling for variable weights, as described in UCA, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• non-ignorable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• shifted</td>
</tr>
<tr>
<td>colBackwards</td>
<td>F</td>
<td>For Latin with accents, this parameter determines which accents are sorted. It sets the comparison for the second level to be backwards.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: colBackwards is automatically set for French accents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set colBackwards to one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• off</td>
</tr>
<tr>
<td>colNormalization</td>
<td>N</td>
<td>Set to one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• off</td>
</tr>
<tr>
<td>colCaseLevel</td>
<td>E</td>
<td>Set to one of the following:</td>
</tr>
<tr>
<td>Parameter</td>
<td>Short form</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **off</td>
</tr>
<tr>
<td>colCaseFirst</td>
<td>C</td>
<td>Set to one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **upper</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **lower</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **off</td>
</tr>
<tr>
<td>colHiraganaQuaternary</td>
<td>H</td>
<td>Controls special treatment of Hiragana code points on quaternary level, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• **off</td>
</tr>
<tr>
<td>colNumeric</td>
<td>D</td>
<td>If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, A-21 &lt; A-123.</td>
</tr>
<tr>
<td>variableTop</td>
<td>B</td>
<td>Sets the default value for the variable top. All code points with primary weights less than or equal to the variable top will be considered variable, and are affected by the alternate...</td>
</tr>
<tr>
<td>Parameter</td>
<td>Short form</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>handling.</td>
<td></td>
<td>For example, the following command sets variableTop to be HYPHEN (u2010)</td>
</tr>
</tbody>
</table>

```
=> \locale en_US@colalternate=shifted;variabletop=u2010
```

### Locale Processing Notes

- Incorrect locale strings are accepted if the prefix can be resolved to a known locale version.

For example, the following works because the language can be resolved:

```
=> \locale en.XX
INFO 2567: Canonical locale: 'en.XX'
Standard collation: 'LEN'
English (XX)
```

The following does not work because the language cannot be resolved:

```
=> \locale xx.XX
xx.XX: invalid locale identifier
```

- POSIX-type locales such as en_US.UTF-8 work to some extent in that the encoding part "UTF-8" is ignored.

- Vertica uses the icu4c-4_2_1 library to support basic locale/collation processing with some extensions. This does not currently meet current standards for locale processing ([https://tools.ietf.org/html/rfc5646](https://tools.ietf.org/html/rfc5646)).

### Examples

Specify German locale as used in Germany (de), with phonebook-style collation:

```
=> \locale de_DE@collation=phonebook
INFO 2567: Canonical locale: 'de_DE@collation=phonebook'
Standard collation: 'KPHONEBOOK_LDE'
German (Germany, collation=Phonebook Sort Order)
Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
```
Specify German locale as used in Germany (de), with phonebook-style collation and strength set to secondary:

```sql
=> \locale de_DE@collation=phonebook;colStrength=secondary
INFO 2567: Canonical locale: 'de_DE@collation=phonebook'
Standard collation: 'KPHONEBOOK_LDE_S2'
German (Germany, collation=Phonebook Sort Order)
Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)
```

**Specifying Locale: Short Form**

Vertica accepts locales in short form. You can use the short form to specify the locale and keyname pair/value names.

To determine the short form for a locale, type in the long form and view the last line of INFO, as follows:

```sql
\locale frINFO: Locale: 'fr'
INFO: French
INFO: français
INFO: Short form: 'LFR'
```

**Examples**

Specify en (English) locale:

```sql
\locale LENINFO: Locale: 'en'
INFO: English
INFO: Short form: 'LEN'
```

Specify German locale as used in Germany (de), with phonebook-style collation:

```sql
\locale LDE_KPHONEBOOKINFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE'
```

Specify German locale as used in Germany (de), with phonebook-style collation:

```sql
\locale LDE_KPHONEBOOK_S2INFO: Locale: 'de@collation=phonebook'
INFO: German (collation=Phonebook Sort Order)
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)
INFO: Short form: 'KPHONEBOOK_LDE_S2'
```
Supported Locales

The following are the supported locale strings for Vertica. Each locale can optionally have a list of key/value pairs (see Specifying Locale: Long Form).

<table>
<thead>
<tr>
<th>Locale Name</th>
<th>Language or Variant</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td></td>
</tr>
<tr>
<td>af_NA</td>
<td>Afrikaans</td>
<td>Namibian Afrikaans</td>
</tr>
<tr>
<td>af_ZA</td>
<td>Afrikaans</td>
<td>South Africa</td>
</tr>
<tr>
<td>am</td>
<td>Ethiopic</td>
<td></td>
</tr>
<tr>
<td>am_ET</td>
<td>Ethiopic</td>
<td>Ethiopia</td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td></td>
</tr>
<tr>
<td>ar_AE</td>
<td>Arabic</td>
<td>United Arab Emirates</td>
</tr>
<tr>
<td>ar_BH</td>
<td>Arabic</td>
<td>Bahrain</td>
</tr>
<tr>
<td>ar_DZ</td>
<td>Arabic</td>
<td>Algeria</td>
</tr>
<tr>
<td>ar_EG</td>
<td>Arabic</td>
<td>Egypt</td>
</tr>
<tr>
<td>ar_IQ</td>
<td>Arabic</td>
<td>Iraq</td>
</tr>
<tr>
<td>ar_JO</td>
<td>Arabic</td>
<td>Jordan</td>
</tr>
<tr>
<td>ar_KW</td>
<td>Arabic</td>
<td>Kuwait</td>
</tr>
<tr>
<td>ar_LB</td>
<td>Arabic</td>
<td>Lebanon</td>
</tr>
<tr>
<td>ar_LY</td>
<td>Arabic</td>
<td>Libya</td>
</tr>
<tr>
<td>ar_MA</td>
<td>Arabic</td>
<td>Morocco</td>
</tr>
<tr>
<td>ar_OM</td>
<td>Arabic</td>
<td>Oman</td>
</tr>
<tr>
<td>ar_QA</td>
<td>Arabic</td>
<td>Qatar</td>
</tr>
<tr>
<td>ar_SA</td>
<td>Arabic</td>
<td>Saudi Arabia</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ar_SD</td>
<td>Arabic</td>
<td>Sudan</td>
</tr>
<tr>
<td>ar_SY</td>
<td>Arabic</td>
<td>Syria</td>
</tr>
<tr>
<td>ar_TN</td>
<td>Arabic</td>
<td>Tunisia</td>
</tr>
<tr>
<td>ar_YE</td>
<td>Arabic</td>
<td>Yemen</td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td></td>
</tr>
<tr>
<td>as_IN</td>
<td>Assamese</td>
<td>India</td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td></td>
</tr>
<tr>
<td>az_Cyrl</td>
<td>Azerbaijani</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>az_Cyrl_AZ</td>
<td>Azerbaijani</td>
<td>Azerbaijan Cyrillic</td>
</tr>
<tr>
<td>az_Latn</td>
<td>Azerbaijani</td>
<td>Latin</td>
</tr>
<tr>
<td>az_Latn_AZ</td>
<td>Azerbaijani</td>
<td>Azerbaijan Latin</td>
</tr>
<tr>
<td>be</td>
<td>Belarusian</td>
<td></td>
</tr>
<tr>
<td>be_BY</td>
<td>Belarusian</td>
<td>Belarus</td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td></td>
</tr>
<tr>
<td>bg_BG</td>
<td>Bulgarian</td>
<td>Bulgaria</td>
</tr>
<tr>
<td>bn</td>
<td>Bengali</td>
<td></td>
</tr>
<tr>
<td>bn_BD</td>
<td>Bengali</td>
<td>Bangladesh</td>
</tr>
<tr>
<td>bn_IN</td>
<td>Bengali</td>
<td>India</td>
</tr>
<tr>
<td>bo</td>
<td>Tibetan</td>
<td></td>
</tr>
<tr>
<td>bo_CN</td>
<td>Tibetan</td>
<td>PR China</td>
</tr>
<tr>
<td>bo_IN</td>
<td>Tibetan</td>
<td>India</td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td></td>
</tr>
<tr>
<td>ca_ES</td>
<td>Catalan</td>
<td>Spain</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>cs</td>
<td>Czech</td>
<td></td>
</tr>
<tr>
<td>cs_CZ</td>
<td>Czech</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cy</td>
<td>Welsh</td>
<td></td>
</tr>
<tr>
<td>cy_GB</td>
<td>Welsh</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>da</td>
<td>Danish</td>
<td></td>
</tr>
<tr>
<td>da_DK</td>
<td>Danish</td>
<td>Denmark</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
<td></td>
</tr>
<tr>
<td>de_AT</td>
<td>German</td>
<td>Austria</td>
</tr>
<tr>
<td>de_BE</td>
<td>German</td>
<td>Belgium</td>
</tr>
<tr>
<td>de_CH</td>
<td>German</td>
<td>Switzerland</td>
</tr>
<tr>
<td>de_DE</td>
<td>German</td>
<td>Germany</td>
</tr>
<tr>
<td>de_LI</td>
<td>German</td>
<td>Liechtenstein</td>
</tr>
<tr>
<td>de_LU</td>
<td>German</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>el</td>
<td>Greek</td>
<td></td>
</tr>
<tr>
<td>el_CY</td>
<td>Greek</td>
<td>Cyprus</td>
</tr>
<tr>
<td>el_GR</td>
<td>Greek</td>
<td>Greece</td>
</tr>
<tr>
<td>en</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td>en_AU</td>
<td>English</td>
<td>Australia</td>
</tr>
<tr>
<td>en_BE</td>
<td>English</td>
<td>Belgium</td>
</tr>
<tr>
<td>en_BW</td>
<td>English</td>
<td>Botswana</td>
</tr>
<tr>
<td>en_BZ</td>
<td>English</td>
<td>Belize</td>
</tr>
<tr>
<td>en_CA</td>
<td>English</td>
<td>Canada</td>
</tr>
<tr>
<td>en_GB</td>
<td>English</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>en_HK</td>
<td>English</td>
<td>Hong Kong S.A.R. of China</td>
</tr>
<tr>
<td>en_IE</td>
<td>English</td>
<td>Ireland</td>
</tr>
<tr>
<td>en_IN</td>
<td>English</td>
<td>India</td>
</tr>
<tr>
<td>en_JM</td>
<td>English</td>
<td>Jamaica</td>
</tr>
<tr>
<td>en_MH</td>
<td>English</td>
<td>Marshall Islands</td>
</tr>
<tr>
<td>en_MT</td>
<td>English</td>
<td>Malta</td>
</tr>
<tr>
<td>en_NA</td>
<td>English</td>
<td>Namibia</td>
</tr>
<tr>
<td>en_NZ</td>
<td>English</td>
<td>New Zealand</td>
</tr>
<tr>
<td>en_PH</td>
<td>English</td>
<td>Philippines</td>
</tr>
<tr>
<td>en_PK</td>
<td>English</td>
<td>Pakistan</td>
</tr>
<tr>
<td>en_SG</td>
<td>English</td>
<td>Singapore</td>
</tr>
<tr>
<td>en_TT</td>
<td>English</td>
<td>Trinidad and Tobago</td>
</tr>
<tr>
<td>en_US</td>
<td>English</td>
<td>United States</td>
</tr>
<tr>
<td>en_US_POSIX</td>
<td>English</td>
<td>United States Posix</td>
</tr>
<tr>
<td>en_VI</td>
<td>English</td>
<td>U.S. Virgin Islands</td>
</tr>
<tr>
<td>en_ZA</td>
<td>English</td>
<td>Zimbabwe or South Africa</td>
</tr>
<tr>
<td>en_ZW</td>
<td>English</td>
<td>Zimbabwe</td>
</tr>
<tr>
<td>eo</td>
<td>Esperanto</td>
<td></td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
<td></td>
</tr>
<tr>
<td>es_AR</td>
<td>Spanish</td>
<td>Argentina</td>
</tr>
<tr>
<td>es_BO</td>
<td>Spanish</td>
<td>Bolivia</td>
</tr>
<tr>
<td>es_CL</td>
<td>Spanish</td>
<td>Chile</td>
</tr>
<tr>
<td>es_CO</td>
<td>Spanish</td>
<td>Columbia</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>es_CR</td>
<td>Spanish</td>
<td>Costa Rica</td>
</tr>
<tr>
<td>es_DO</td>
<td>Spanish</td>
<td>Dominican Republic</td>
</tr>
<tr>
<td>es_EC</td>
<td>Spanish</td>
<td>Ecuador</td>
</tr>
<tr>
<td>es_ES</td>
<td>Spanish</td>
<td>Spain</td>
</tr>
<tr>
<td>es_GT</td>
<td>Spanish</td>
<td>Guatemala</td>
</tr>
<tr>
<td>es_HN</td>
<td>Spanish</td>
<td>Honduras</td>
</tr>
<tr>
<td>es_MX</td>
<td>Spanish</td>
<td>Mexico</td>
</tr>
<tr>
<td>es_NI</td>
<td>Spanish</td>
<td>Nicaragua</td>
</tr>
<tr>
<td>es_PA</td>
<td>Spanish</td>
<td>Panama</td>
</tr>
<tr>
<td>es_PE</td>
<td>Spanish</td>
<td>Peru</td>
</tr>
<tr>
<td>es_PR</td>
<td>Spanish</td>
<td>Puerto Rico</td>
</tr>
<tr>
<td>es_PY</td>
<td>Spanish</td>
<td>Paraguay</td>
</tr>
<tr>
<td>es_SV</td>
<td>Spanish</td>
<td>El Salvador</td>
</tr>
<tr>
<td>es_US</td>
<td>Spanish</td>
<td>United States</td>
</tr>
<tr>
<td>es_UY</td>
<td>Spanish</td>
<td>Uruguay</td>
</tr>
<tr>
<td>es_VE</td>
<td>Spanish</td>
<td>Venezuela</td>
</tr>
<tr>
<td>et</td>
<td>Estonian</td>
<td></td>
</tr>
<tr>
<td>et_EE</td>
<td>Estonian</td>
<td>Estonia</td>
</tr>
<tr>
<td>eu</td>
<td>Basque</td>
<td>Spain</td>
</tr>
<tr>
<td>eu_ES</td>
<td>Basque</td>
<td>Spain</td>
</tr>
<tr>
<td>fa</td>
<td>Persian</td>
<td></td>
</tr>
<tr>
<td>fa_AF</td>
<td>Persian</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>fa_IR</td>
<td>Persian</td>
<td>Iran</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
<td></td>
</tr>
<tr>
<td>fi_FI</td>
<td>Finnish</td>
<td>Finland</td>
</tr>
<tr>
<td>fo</td>
<td>Faroese</td>
<td></td>
</tr>
<tr>
<td>fo_FO</td>
<td>Faroese</td>
<td>Faroe Islands</td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
<td></td>
</tr>
<tr>
<td>fr_BE</td>
<td>French</td>
<td>Belgium</td>
</tr>
<tr>
<td>fr_CA</td>
<td>French</td>
<td>Canada</td>
</tr>
<tr>
<td>fr_CH</td>
<td>French</td>
<td>Switzerland</td>
</tr>
<tr>
<td>fr_FR</td>
<td>French</td>
<td>France</td>
</tr>
<tr>
<td>fr_LU</td>
<td>French</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>fr_MC</td>
<td>French</td>
<td>Monaco</td>
</tr>
<tr>
<td>fr_SN</td>
<td>French</td>
<td>Senegal</td>
</tr>
<tr>
<td>ga</td>
<td>Gaelic</td>
<td></td>
</tr>
<tr>
<td>ga_IE</td>
<td>Gaelic</td>
<td>Ireland</td>
</tr>
<tr>
<td>gl</td>
<td>Gallegan</td>
<td></td>
</tr>
<tr>
<td>gl_ES</td>
<td>Gallegan</td>
<td>Spain</td>
</tr>
<tr>
<td>gsw</td>
<td>German</td>
<td></td>
</tr>
<tr>
<td>gsw_CH</td>
<td>German</td>
<td>Switzerland</td>
</tr>
<tr>
<td>gu</td>
<td>Gujurati</td>
<td></td>
</tr>
<tr>
<td>gu_IN</td>
<td>Gujurati</td>
<td>India</td>
</tr>
<tr>
<td>gv</td>
<td>Manx</td>
<td></td>
</tr>
<tr>
<td>gv_GB</td>
<td>Manx</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>ha</td>
<td>Hausa</td>
<td></td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>ha_Latn</td>
<td>Hausa</td>
<td>Latin</td>
</tr>
<tr>
<td>ha_Latn_GH</td>
<td>Hausa</td>
<td>Ghana (Latin)</td>
</tr>
<tr>
<td>ha_Latn_NE</td>
<td>Hausa</td>
<td>Niger (Latin)</td>
</tr>
<tr>
<td>ha_Latn_NG</td>
<td>Hausa</td>
<td>Nigeria (Latin)</td>
</tr>
<tr>
<td>haw</td>
<td>Hawaiian</td>
<td>Hawaiian</td>
</tr>
<tr>
<td>haw_US</td>
<td>Hawaiian</td>
<td>United States</td>
</tr>
<tr>
<td>he</td>
<td>Hebrew</td>
<td></td>
</tr>
<tr>
<td>he_IL</td>
<td>Hebrew</td>
<td>Israel</td>
</tr>
<tr>
<td>hi</td>
<td>Hindi</td>
<td></td>
</tr>
<tr>
<td>hi_IN</td>
<td>Hindi</td>
<td>India</td>
</tr>
<tr>
<td>hr</td>
<td>Croatian</td>
<td></td>
</tr>
<tr>
<td>hr_HR</td>
<td>Croatian</td>
<td>Croatia</td>
</tr>
<tr>
<td>hu</td>
<td>Hungarian</td>
<td></td>
</tr>
<tr>
<td>hu_HU</td>
<td>Hungarian</td>
<td>Hungary</td>
</tr>
<tr>
<td>hy</td>
<td>Armenian</td>
<td></td>
</tr>
<tr>
<td>hy_AM</td>
<td>Armenian</td>
<td>Armenia</td>
</tr>
<tr>
<td>hy_AM_REVISED</td>
<td>Armenian</td>
<td>Revised Armenia</td>
</tr>
<tr>
<td>id</td>
<td>Indonesian</td>
<td></td>
</tr>
<tr>
<td>id_ID</td>
<td>Indonesian</td>
<td>Indonesia</td>
</tr>
<tr>
<td>ii</td>
<td>Sichuan</td>
<td></td>
</tr>
<tr>
<td>ii_CN</td>
<td>Sichuan</td>
<td>Yi</td>
</tr>
<tr>
<td>is</td>
<td>Icelandic</td>
<td></td>
</tr>
<tr>
<td>is_IS</td>
<td>Icelandic</td>
<td>Iceland</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
<td></td>
</tr>
<tr>
<td>it_CH</td>
<td>Italian</td>
<td>Switzerland</td>
</tr>
<tr>
<td>it_IT</td>
<td>Italian</td>
<td>Italy</td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
<td></td>
</tr>
<tr>
<td>ja_JP</td>
<td>Japanese</td>
<td>Japan</td>
</tr>
<tr>
<td>ka</td>
<td>Georgian</td>
<td></td>
</tr>
<tr>
<td>ka_GE</td>
<td>Georgian</td>
<td>Georgia</td>
</tr>
<tr>
<td>kk</td>
<td>Kazakh</td>
<td></td>
</tr>
<tr>
<td>kk_Cyrl</td>
<td>Kazakh</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>kk_Cyrl_KZ</td>
<td>Kazakh</td>
<td>Kazakhstan (Cyrillic)</td>
</tr>
<tr>
<td>kl</td>
<td>Kalaallisut</td>
<td></td>
</tr>
<tr>
<td>kl_GL</td>
<td>Kalaallisut</td>
<td>Greenland</td>
</tr>
<tr>
<td>km</td>
<td>Khmer</td>
<td></td>
</tr>
<tr>
<td>km_KH</td>
<td>Khmer</td>
<td>Cambodia</td>
</tr>
<tr>
<td>kn</td>
<td>Kannada</td>
<td></td>
</tr>
<tr>
<td>kn-IN</td>
<td>Kannada</td>
<td>India</td>
</tr>
<tr>
<td>ko</td>
<td>Korean</td>
<td></td>
</tr>
<tr>
<td>ko_KR</td>
<td>Korean</td>
<td>Korea</td>
</tr>
<tr>
<td>kok</td>
<td>Konkani</td>
<td></td>
</tr>
<tr>
<td>kok_IN</td>
<td>Konkani</td>
<td>India</td>
</tr>
<tr>
<td>kw</td>
<td>Cornish</td>
<td></td>
</tr>
<tr>
<td>kw_GB</td>
<td>Cornish</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>lt</td>
<td>Lithuanian</td>
<td></td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>------------</td>
</tr>
<tr>
<td>lt_LT</td>
<td>Lithuanian</td>
<td>Lithuania</td>
</tr>
<tr>
<td>lv</td>
<td>Latvian</td>
<td></td>
</tr>
<tr>
<td>lv_LV</td>
<td>Latvian</td>
<td>Latvia</td>
</tr>
<tr>
<td>mk</td>
<td>Macedonian</td>
<td></td>
</tr>
<tr>
<td>mk_MK</td>
<td>Macedonian</td>
<td>Macedonia</td>
</tr>
<tr>
<td>ml</td>
<td>Malayalam</td>
<td></td>
</tr>
<tr>
<td>ml_IN</td>
<td>Malayalam</td>
<td>India</td>
</tr>
<tr>
<td>mr</td>
<td>Marathi</td>
<td></td>
</tr>
<tr>
<td>mr_IN</td>
<td>Marathi</td>
<td>India</td>
</tr>
<tr>
<td>ms</td>
<td>Malay</td>
<td></td>
</tr>
<tr>
<td>ms_BN</td>
<td>Malay</td>
<td>Brunei</td>
</tr>
<tr>
<td>ms_MY</td>
<td>Malay</td>
<td>Malaysia</td>
</tr>
<tr>
<td>mt</td>
<td>Maltese</td>
<td></td>
</tr>
<tr>
<td>mt_MT</td>
<td>Maltese</td>
<td>Malta</td>
</tr>
<tr>
<td>nb</td>
<td>Norwegian Bokml</td>
<td></td>
</tr>
<tr>
<td>nb_NO</td>
<td>Norwegian Bokml</td>
<td>Norway</td>
</tr>
<tr>
<td>ne</td>
<td>Nepali</td>
<td></td>
</tr>
<tr>
<td>ne_IN</td>
<td>Nepali</td>
<td>India</td>
</tr>
<tr>
<td>ne_NP</td>
<td>Nepali</td>
<td>Nepal</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
<td></td>
</tr>
<tr>
<td>nl_BE</td>
<td>Dutch</td>
<td>Belgium</td>
</tr>
<tr>
<td>nl_NL</td>
<td>Dutch</td>
<td>Netherlands</td>
</tr>
<tr>
<td>nn</td>
<td>Norwegian nynorsk</td>
<td></td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>nn_NO</td>
<td>Norwegian nynorsk</td>
<td>Norway</td>
</tr>
<tr>
<td>om</td>
<td>Oromo</td>
<td></td>
</tr>
<tr>
<td>om_ET</td>
<td>Oromo</td>
<td>Ethiopia</td>
</tr>
<tr>
<td>om KE</td>
<td>Oromo</td>
<td>Kenya</td>
</tr>
<tr>
<td>or</td>
<td>Oriya</td>
<td></td>
</tr>
<tr>
<td>or_IN</td>
<td>Oriya</td>
<td>India</td>
</tr>
<tr>
<td>pa</td>
<td>Punjabi</td>
<td></td>
</tr>
<tr>
<td>pa_Arab</td>
<td>Punjabi</td>
<td>Arabic</td>
</tr>
<tr>
<td>pa_Arab_PK</td>
<td>Punjabi</td>
<td>Pakistan (Arabic)</td>
</tr>
<tr>
<td>pa_Guru</td>
<td>Punjabi</td>
<td>Gurmukhi</td>
</tr>
<tr>
<td>pa_Guru_IN</td>
<td>Punjabi</td>
<td>India (Gurmukhi)</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
<td></td>
</tr>
<tr>
<td>pl_PL</td>
<td>Polish</td>
<td>Poland</td>
</tr>
<tr>
<td>ps</td>
<td>Pashto</td>
<td></td>
</tr>
<tr>
<td>ps_AF</td>
<td>Pashto</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
<td></td>
</tr>
<tr>
<td>pt_BR</td>
<td>Portuguese</td>
<td>Brazil</td>
</tr>
<tr>
<td>pt_PT</td>
<td>Portuguese</td>
<td>Portugal</td>
</tr>
<tr>
<td>ro</td>
<td>Romanian</td>
<td></td>
</tr>
<tr>
<td>ro_MD</td>
<td>Romanian</td>
<td>Moldavia</td>
</tr>
<tr>
<td>ro_RO</td>
<td>Romanian</td>
<td>Romania</td>
</tr>
<tr>
<td>ru</td>
<td>Russian</td>
<td></td>
</tr>
<tr>
<td>ru_RU</td>
<td>Russian</td>
<td>Russia</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>ru_UA</td>
<td>Russian</td>
<td>Ukraine</td>
</tr>
<tr>
<td>si</td>
<td>Sinhala</td>
<td></td>
</tr>
<tr>
<td>si_LK</td>
<td>Sinhala</td>
<td>Sri Lanka</td>
</tr>
<tr>
<td>sk</td>
<td>Slovak</td>
<td></td>
</tr>
<tr>
<td>sk.SK</td>
<td>Slovak</td>
<td>Slovakia</td>
</tr>
<tr>
<td>sl</td>
<td>Slovenian</td>
<td></td>
</tr>
<tr>
<td>sl_SL</td>
<td>Slovenian</td>
<td>Slovenia</td>
</tr>
<tr>
<td>so</td>
<td>Somali</td>
<td></td>
</tr>
<tr>
<td>so_DJ</td>
<td>Somali</td>
<td>Djibouti</td>
</tr>
<tr>
<td>so_ET</td>
<td>Somali</td>
<td>Ethiopia</td>
</tr>
<tr>
<td>so_KG</td>
<td>Somali</td>
<td>Kenya</td>
</tr>
<tr>
<td>so_SO</td>
<td>Somali</td>
<td>Somalia</td>
</tr>
<tr>
<td>sq</td>
<td>Albanian</td>
<td></td>
</tr>
<tr>
<td>sq_AL</td>
<td>Albanian</td>
<td>Albania</td>
</tr>
<tr>
<td>sr</td>
<td>Serbian</td>
<td></td>
</tr>
<tr>
<td>sr_Cyrl</td>
<td>Serbian</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>sr_Cyrl_BA</td>
<td>Serbian</td>
<td>Bosnia and Herzegovina</td>
</tr>
<tr>
<td>sr_Cyrl_ME</td>
<td>Serbian</td>
<td>Montenegro</td>
</tr>
<tr>
<td>sr_Cyrl_RS</td>
<td>Serbian</td>
<td>Serbia</td>
</tr>
<tr>
<td>sr_Latn</td>
<td>Serbian</td>
<td>Latin</td>
</tr>
<tr>
<td>sr_Latn_BA</td>
<td>Serbian</td>
<td>Bosnia and Herzegovina</td>
</tr>
<tr>
<td>sr_Latn_ME</td>
<td>Serbian</td>
<td>Montenegro</td>
</tr>
<tr>
<td>sr_Latn_RS</td>
<td>Serbian</td>
<td>Serbia</td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>sv</td>
<td>Swedish</td>
<td></td>
</tr>
<tr>
<td>sv_FI</td>
<td>Swedish</td>
<td>Finland</td>
</tr>
<tr>
<td>sv_SE</td>
<td>Swedish</td>
<td>Sweden</td>
</tr>
<tr>
<td>sw</td>
<td>Swahili</td>
<td></td>
</tr>
<tr>
<td>sw KE</td>
<td>Swahili</td>
<td>Kenya</td>
</tr>
<tr>
<td>sw_TZ</td>
<td>Swahili</td>
<td>Tanzania</td>
</tr>
<tr>
<td>ta</td>
<td>Tamil</td>
<td></td>
</tr>
<tr>
<td>ta_IN</td>
<td>Tamil</td>
<td>India</td>
</tr>
<tr>
<td>te</td>
<td>Telugu</td>
<td></td>
</tr>
<tr>
<td>te_IN</td>
<td>Telugu</td>
<td>India</td>
</tr>
<tr>
<td>th</td>
<td>Thai</td>
<td></td>
</tr>
<tr>
<td>th_TH</td>
<td>Thai</td>
<td>Thailand</td>
</tr>
<tr>
<td>ti</td>
<td>Tigrinya</td>
<td></td>
</tr>
<tr>
<td>ti_ER</td>
<td>Tigrinya</td>
<td>Eritrea</td>
</tr>
<tr>
<td>ti_ET</td>
<td>Tigrinya</td>
<td>Ethiopia</td>
</tr>
<tr>
<td>tr</td>
<td>Turkish</td>
<td></td>
</tr>
<tr>
<td>tr_TR</td>
<td>Turkish</td>
<td>Turkey</td>
</tr>
<tr>
<td>uk</td>
<td>Ukrainian</td>
<td></td>
</tr>
<tr>
<td>uk_UA</td>
<td>Ukrainian</td>
<td>Ukraine</td>
</tr>
<tr>
<td>ur</td>
<td>Urdu</td>
<td></td>
</tr>
<tr>
<td>ur_IN</td>
<td>Urdu</td>
<td>India</td>
</tr>
<tr>
<td>ur_PK</td>
<td>Urdu</td>
<td>Pakistan</td>
</tr>
<tr>
<td>uz</td>
<td>Uzbek</td>
<td></td>
</tr>
<tr>
<td>Locale Name</td>
<td>Language or Variant</td>
<td>Region</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>uz_Arab</td>
<td>Uzbek</td>
<td>Arabic</td>
</tr>
<tr>
<td>uz_Arab_AF</td>
<td>Uzbek</td>
<td>Afghanistan (Arabic)</td>
</tr>
<tr>
<td>uz_Cryl</td>
<td>Uzbek</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>uz_Cryl_UZ</td>
<td>Uzbek</td>
<td>Uzbekistan (Cyrillic)</td>
</tr>
<tr>
<td>uz_Latin</td>
<td>Uzbek</td>
<td>Latin</td>
</tr>
<tr>
<td>us_Latin_UZ</td>
<td></td>
<td>Uzbekistan (Latin)</td>
</tr>
<tr>
<td>vi</td>
<td>Vietnamese</td>
<td></td>
</tr>
<tr>
<td>vi_VN</td>
<td>Vietnamese</td>
<td>Vietnam</td>
</tr>
<tr>
<td>zh</td>
<td>Chinese</td>
<td></td>
</tr>
<tr>
<td>zh_Hans</td>
<td>Chinese</td>
<td>Simplified Han</td>
</tr>
<tr>
<td>zh_Hans_CN</td>
<td>Chinese</td>
<td>China (Simplified Han)</td>
</tr>
<tr>
<td>zh_Hans_HK</td>
<td>Chinese</td>
<td>Hong Kong SAR China (Simplified Han)</td>
</tr>
<tr>
<td>zh_Hans_MO</td>
<td>Chinese</td>
<td>Macao SAR China (Simplified Han)</td>
</tr>
<tr>
<td>zh_Hans_SG</td>
<td>Chinese</td>
<td>Singapore (Simplified Han)</td>
</tr>
<tr>
<td>zh_Hant</td>
<td>Chinese</td>
<td>Traditional Han</td>
</tr>
<tr>
<td>zh_Hant_HK</td>
<td>Chinese</td>
<td>Hong Kong SAR China (Traditional Han)</td>
</tr>
<tr>
<td>zh_Hant_MO</td>
<td>Chinese</td>
<td>Macao SAR China (Traditional Han)</td>
</tr>
<tr>
<td>zh_Hant_TW</td>
<td>Chinese</td>
<td>Taiwan (Traditional Han)</td>
</tr>
<tr>
<td>zu</td>
<td>Zulu</td>
<td></td>
</tr>
<tr>
<td>zu_ZA</td>
<td>Zulu</td>
<td>South Africa</td>
</tr>
</tbody>
</table>
Locale and UTF-8 Support

Vertica supports Unicode Transformation Format-8, or UTF8, where 8 equals 8-bit. UTF-8 is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard. Initial encoding of byte codes and character assignments for UTF-8 coincides with ASCII. Thus, UTF8 requires little or no change for software that handles ASCII but preserves other values.

Vertica database servers expect to receive all data in UTF-8, and Vertica outputs all data in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. JDBC and ADO.NET APIs operate on data in UTF-16. Client drivers automatically convert data to and from UTF-8 when sending to and receiving data from Vertica using API calls. The drivers do not transform data loaded by executing a COPY or COPY LOCAL statement.

UTF-8 String Functions

The following string functions treat VARCHAR arguments as UTF-8 strings (when USING OCTETS is not specified) regardless of locale setting.

<table>
<thead>
<tr>
<th>String function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER</td>
<td>Returns a VARCHAR value containing the argument converted to lowercase letters.</td>
</tr>
<tr>
<td>UPPER</td>
<td>Returns a VARCHAR value containing the argument converted to uppercase letters.</td>
</tr>
<tr>
<td>INITCAP</td>
<td>Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.</td>
</tr>
<tr>
<td>INSTR</td>
<td>Searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.</td>
</tr>
<tr>
<td>SPLIT_PART</td>
<td>Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).</td>
</tr>
<tr>
<td>POSITION</td>
<td>Returns an integer value representing the character location of a specified substring with a string (counting from one).</td>
</tr>
<tr>
<td>String function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>STRPOS</td>
<td>Returns an integer value representing the character location of a specified substring within a string (counting from one).</td>
</tr>
</tbody>
</table>

## Locale-Aware String Functions

Vertica provides string functions to support internationalization. Unless otherwise specified, these string functions can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of characters. Specify this information by adding the parameter USING OCTETS and USING CHARACTERS (default) to the function.

The following table lists all string functions that are locale-aware:

<table>
<thead>
<tr>
<th>String function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTRIM</td>
<td>Removes the longest string consisting only of specified characters from the start and end of a string.</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>Returns an integer value representing the number of characters or octets in a string.</td>
</tr>
<tr>
<td>GREATEST</td>
<td>Returns the largest value in a list of expressions.</td>
</tr>
<tr>
<td>GREATESTB</td>
<td>Returns its greatest argument, using binary ordering, not UTF-8 character ordering.</td>
</tr>
<tr>
<td>INITCAP</td>
<td>Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.</td>
</tr>
<tr>
<td>INSTR</td>
<td>Searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.</td>
</tr>
<tr>
<td>LEAST</td>
<td>Returns the smallest value in a list of expressions.</td>
</tr>
<tr>
<td>LEASTB</td>
<td>Returns its least argument, using binary ordering, not UTF-8 character ordering.</td>
</tr>
<tr>
<td>LEFT</td>
<td>Returns the specified characters from the left side of a string.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>Takes one argument as an input and returns an integer value</td>
</tr>
<tr>
<td>String function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>representing the number of characters in a string.</td>
</tr>
<tr>
<td><strong>LTRIM</strong></td>
<td>Returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).</td>
</tr>
<tr>
<td><strong>OVERLAY</strong></td>
<td>Returns a VARCHAR value representing a string having had a substring replaced by another string.</td>
</tr>
<tr>
<td><strong>OVERLAYB</strong></td>
<td>Returns an octet value representing a string having had a substring replaced by another string.</td>
</tr>
<tr>
<td><strong>REPLACE</strong></td>
<td>replaces all occurrences of characters in a string with another set of characters.</td>
</tr>
<tr>
<td><strong>RIGHT</strong></td>
<td>Returns the <em>length</em> right-most characters of string.</td>
</tr>
<tr>
<td><strong>SUBSTR</strong></td>
<td>Returns a VARCHAR value representing a substring of a specified string.</td>
</tr>
<tr>
<td><strong>SUBSTRB</strong></td>
<td>Returns a byte value representing a substring of a specified string.</td>
</tr>
<tr>
<td><strong>SUBSTRING</strong></td>
<td>Given a value, a position, and an optional length, returns a value representing a substring of the specified string at the given position.</td>
</tr>
<tr>
<td><strong>TRANSLATE</strong></td>
<td>Replaces individual characters in <em>string_to_replace</em> with other characters.</td>
</tr>
<tr>
<td><strong>UPPER</strong></td>
<td>Returns a VARCHAR value containing the argument converted to uppercase letters.</td>
</tr>
</tbody>
</table>
Appendix: Creating Native Binary Format Files

Using COPY to load data with the NATIVE parser requires that the input data files conform to the requirements described in this appendix. All NATIVE files must contain:

- File signature
- Column size definitions
- Rows of data

**Note:** You cannot mix Binary and ASCII source files in the same COPY statement.

### File Signature

The first part of a NATIVE binary file consists of a file signature. The contents of the signature are fixed, and listed in the following table.

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex Value</td>
<td>4E</td>
<td>41</td>
<td>54</td>
<td>49</td>
<td>56</td>
<td>45</td>
<td>0A</td>
<td>FF</td>
<td>0D</td>
<td>0A</td>
<td>00</td>
</tr>
<tr>
<td>Text Literals</td>
<td>N</td>
<td>A</td>
<td>T</td>
<td>I</td>
<td>V</td>
<td>E</td>
<td>E \n</td>
<td>E \317</td>
<td>E \n</td>
<td>E \n</td>
<td>E \00</td>
</tr>
</tbody>
</table>

The signature ensures that the file has neither been corrupted by a non-8-bit file transfer, nor stripped of carriage returns, linefeeds, or null values. If the signature is intact, Vertica determines that the file has not been corrupted.

### Column Definitions

Following the file signature, the file must define the widths of each column in the file as follows.
### Byte Offset | Length (bytes) | Description | Comments
---|---|---|---
11 | 4 | Header area length | 32-bit integer in little-endian format that contains the length in bytes of remaining in the header, not including itself. This is the number of bytes from the end of this value to the start of the row data.
15 | 2 | NATIVE file version | 16-bit integer in little-endian format containing the version number of the NATIVE file format. The only valid value is currently 1. Future changes to the format could be assigned different version numbers to maintain backward compatibility.
17 | 1 | Filler | Always 0.
18 | 2 | Number of columns | 16-bit integer in little-endian format that contains the number of columns in each row in the file.
20+ | 4 bytes for each column of data in the table | Column widths | Array of 32-bit integers in little-endian format that define the width of each column in the row. Variable-width columns have a value of -1 (0xFF 0xFF 0xFF 0xFF).

**Note:** All integers in NATIVE files are in little-endian format (least significant byte first).

The width of each column is determined by the data type it contains. The following table explains the column width needed for each data type, along with the data encoding.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Column Content</th>
</tr>
</thead>
</table>
| INTEGER | 1, 2, 4, 8 | 8-, 16-, 32-, and 64-bit integers are supported. All multi-byte values are stored in little-endian format.  
**Note:** All values for a column must be the width you specify here. If you set the length of an INTEGER column to be 4 bytes, then all of the values you supply for that column must be 32-bit integers.  
| BOOLEAN | 1 | 0 for false, 1 for true.  
| FLOAT | 8 | Encoded in IEEE-754 format.  

---

Vertica Analytic Database (9.0.x)  
Page 1414 of 6180
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Column Content</th>
</tr>
</thead>
</table>
| CHAR      | User-specified | - Strings shorter than the specified length must be right-padded with spaces (E'\040').  
- Strings are not null-terminated.  
- Character encoding is UTF-8.  
- UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes. |
| VARCHAR   | 4-byte integer (length) + data | The column width for a VARCHAR column is always -1 to signal that it contains variable-length data.  
- Each VARCHAR column value starts with a 32-bit integer that contains the number of bytes in the string.  
- The string must not be null-terminated.  
- Character encoding must be UTF-8.  
- Remember that UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes. |
| DATE      | 8              | 64-bit integer in little-endian format containing the Julian day since Jan 01 2000 (J2451545) |
| TIME      | 8              | 64-bit integer in little-endian format containing the number of microseconds since midnight in the UTC time zone. |
| TIMETZ    | 8              | 64-bit value where  
- Upper 40 bits contain the number of microseconds since midnight.  
- Lower 24 bits contain time zone as the UTC offset in microseconds calculated as follows: Time zone is logically from -24hrs to +24hrs from UTC. Instead it is represented here as a number between 0hrs to 48hrs. Therefore, 24hrs should be added to the actual time |
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Length (bytes)</th>
<th>Column Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>zone to calculate it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each portion is stored in little-endian format (5 bytes followed by 3 bytes).</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>8</td>
<td>64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00.</td>
</tr>
<tr>
<td>TIMESTAMPTZ</td>
<td>8</td>
<td>A 64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00 in the UTC timezone.</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>8</td>
<td>64-bit integer in little-endian format containing the number of microseconds in the interval.</td>
</tr>
<tr>
<td>BINARY</td>
<td>User-specified</td>
<td>Similar to CHAR. The length should be specified in the file header in the Field Lengths entry for the field. The field in the record must contain length number of bytes. If the value is smaller than the specified length, the remainder should be filled with nulls (E'\000').</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>4-byte integer + data</td>
<td>Stored just like VARCHAR but data is interpreted as bytes rather than UTF-8 characters.</td>
</tr>
</tbody>
</table>
| NUMERIC       | (precision, scale) | (precision ÷ 19 + 1) × 8 rounded up | A constant-length data type. Length is determined by the precision, assuming that a 64-bit unsigned integer can store roughly 19 decimal digits. The data consists of a sequence of 64-bit integers, each stored in little-endian format, with the most significant integer first. Data in the integers is stored in base 2^{64}. 2's complement is used for negative numbers.
If there is a scale, then the numeric is stored as numeric × 10^{scale}; that is, all real numbers are stored as integers, ignoring the decimal point. It is required that the scale matches that of the target column in the dataanchor table. Another option is to use FILLER columns to coerce the numeric to the scale of the target column.
# Row Data

Following the file header is a sequence of records that contain the data for each row of data. Each record starts with a header:

<table>
<thead>
<tr>
<th>Length (bytes)</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Row length</td>
<td>A 32-bit integer in little-endian format containing the length of the row's data in bytes. It includes the size of data only, not the header.</td>
</tr>
</tbody>
</table>

**Note:** The number of bytes in each row can vary not only because of variable-length data, but also because columns containing NULL values do not have any data in the row. If column 3 has a NULL value, then column 4's data immediately follows the end of column 2's data. See the next

| Number of columns ÷ 8 rounded up (CEILING (NumFields / ( sizeof (uint8) * 8) )); | Null value bit field | A series of bytes whose bits indicate whether a column contains a NULL. The most significant bit of the first byte indicates whether the first column in this row contains a NULL, the next most significant bit indicates whether the next column contains a NULL, and so on. If a bit is 1 (true) then the column contains a NULL, and there is no value for the column in the data for the row. |

Following the record header is the column values for the row. There is no separator characters for these values. Their location in the row of data is calculated based on where the previous column's data ended. Most data types have a fixed width, so their location is easy to determine. Variable-width values (such as VARCHAR and VARBINARY) start with a count of the number of bytes the value contains.

See the table in the previous section for details on how each data type's value is stored in the row's data.
Example

The example below demonstrates creating a table and loading a NATIVE file that contains a single row of data. The table contains all possible data types.

=> CREATE TABLE allTypes (INTCOL INTEGER, FLOATCOL FLOAT, CHARCOL CHAR(10), VARCHARCOL VARCHAR, BOOLCOL BOOLEAN, DATECOL DATE, TIMESTAMPCOL TIMESTAMP, TIMESTAMPTZCOL TIMESTAMPTZ, TIMECOL TIME, TIMETZCOL TIMETZ, VARBINCOL VARBINARY, BINCOL BINARY, NUMCOL NUMERIC(38,0), INTERVALCOL INTERVAL);
=> COPY allTypes FROM '/home/dbadmin/allTypes.bin' NATIVE DIRECT;
=> \ps expanded
Expanded display is on.
=> SELECT * from allTypes;
-[ RECORD 1 ]-----------------------------------------
| INTCOL | 1 | FLOACOL | -1.11 | CHARCOL | one | VARCHARCOL | ONE | BOOLCOL | t | DATECOL | 1999-01-08 | TIMESTAMPCOL | 1999-02-23 03:11:52.35 | TIMESTAMPTZCOL | 1999-01-08 07:04:37-05 | TIMECOL | 07:09:23 | TIMETZCOL | 15:12:34-04 | VARBINCOL | \253\315 | BINCOL | \253 | NUMCOL | 1234532 | INTERVALCOL | 03:03:03 |

The content of the allTypes.bin file appears below as a raw hex dump:

```
4E 41 54 49 56 45 0A FF 80 0A 00 3D 00 00 00 01 00 00 0E 00
08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF FF 03 00 00 00 18 00 00 00 08 00 00 00 73 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0 2E 98 FF 05 00 00 00 00 97 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB CD AB CD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 64 D6 12 00 00 00 00 00 C0 47 A3 8E 02 00 00 00
```
The following table breaks this file down into each of its components, and describes the values it contains.

<table>
<thead>
<tr>
<th>Hex Values</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4E 41 54 49 56 45 0A FF 0D 0A 00</td>
<td>Signature</td>
<td>NATIVE\n\317\r\n000</td>
</tr>
<tr>
<td>3D 00 00 00</td>
<td>Header area length</td>
<td>61 bytes</td>
</tr>
<tr>
<td>01 00</td>
<td>Native file format version</td>
<td>Version 1</td>
</tr>
<tr>
<td>00</td>
<td>Filler value</td>
<td>0</td>
</tr>
<tr>
<td>0E 00</td>
<td>Number of columns</td>
<td>14 columns</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 1 (INTEGER)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 2 (FLOAT)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>0A 00 00 00</td>
<td>Width of column 3 (CHAR(10))</td>
<td>10 bytes</td>
</tr>
<tr>
<td>FF FF FF FF</td>
<td>Width of column 4 (VARCHAR)</td>
<td>-1 (variable width column)</td>
</tr>
<tr>
<td>01 00 00 00</td>
<td>Width of column 5 (BOOLEAN)</td>
<td>1 bytes</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 6 (DATE)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 7 (TIMESTAMP)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 8 (TIMESTAMPTZ)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 9 (TIME)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Hex Values</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 10 (TIMETZ)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>FF FF FF FF</td>
<td>Width of column 11 (VARBINARY)</td>
<td>-1 (variable width column)</td>
</tr>
<tr>
<td>03 00 00 00</td>
<td>Width of column 12 (BINARY)</td>
<td>3 bytes</td>
</tr>
<tr>
<td>18 00 00 00</td>
<td>Width of column 13 (NUMERIC)</td>
<td>24 bytes. The size is calculated by dividing 38 (the precision specified for the numeric column) by 19 (the number of digits each 64-bit chunk can represent) and adding 1. $38 \div 19 + 1 = 3$. then multiply by eight to get the number of bytes needed. $3 \times 8 = 24$ bytes.</td>
</tr>
<tr>
<td>08 00 00 00</td>
<td>Width of column 14 (INTERVAL). last portion of the header section.</td>
<td>8 bytes</td>
</tr>
<tr>
<td>73 00 00 00</td>
<td>Number of bytes of data for the first row. this is the start of the first row of data.</td>
<td>115 bytes</td>
</tr>
<tr>
<td>00 00</td>
<td>Bit field for the null values contained in the first row of data.</td>
<td>The row contains no null values.</td>
</tr>
<tr>
<td>01 00 00 00</td>
<td>Value for 64-bit INTEGER column</td>
<td>1</td>
</tr>
<tr>
<td>00 00 00 00</td>
<td>Value for the FLOAT column</td>
<td>-1.11</td>
</tr>
<tr>
<td>6F 6E 65 20</td>
<td>Value for the CHAR(10) column</td>
<td>&quot;one &quot; (padded With 7 spaces to fill the full 10 characters for the column)</td>
</tr>
<tr>
<td>Hex Values</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>03 00 00 00</td>
<td>The number of bytes in the following VARCHAR value.</td>
<td>3 bytes</td>
</tr>
<tr>
<td>4F 4E 45</td>
<td>The value for the VARCHAR column</td>
<td>&quot;ONE&quot;</td>
</tr>
<tr>
<td>01</td>
<td>The value for the BOOLEAN column</td>
<td>True</td>
</tr>
<tr>
<td>9A FE FF FF FF FF FF FF FF FF</td>
<td>The value for the DATE column</td>
<td>1999-01-08</td>
</tr>
<tr>
<td>30 85 B3 4F 7E E7 FF FF FF</td>
<td>The value for the TIMESTAMP column</td>
<td>1999-02-23 03:11:52.35</td>
</tr>
<tr>
<td>40 1F 3E 64 E8 E3 FF FF FF</td>
<td>The value for the TIMESTAMPTZ column</td>
<td>1999-01-08 07:04:37-05</td>
</tr>
<tr>
<td>C0 2E 98 FF 05 00 00 00</td>
<td>The value for the TIME column</td>
<td>07:09:23</td>
</tr>
<tr>
<td>D0 97 01 80 F0 79 F0 10</td>
<td>The value for the TIMETZ column</td>
<td>15:12:34-05</td>
</tr>
<tr>
<td>02 00 00 00</td>
<td>The number of bytes in the following VARBINARY value</td>
<td>2 bytes</td>
</tr>
<tr>
<td>AB CD</td>
<td>The value for the VARBINARY column</td>
<td>Binary data (\253\315 as octal values)</td>
</tr>
<tr>
<td>AB CD</td>
<td>The value for the BINARY column</td>
<td>Binary data (\253\315 as octal values)</td>
</tr>
<tr>
<td>00 00 00 00 00 00 00 00</td>
<td>The value for the NUMERIC column</td>
<td>1234532</td>
</tr>
<tr>
<td>Hex Values</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 D6 12 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C0 47 A3 8E</td>
<td>The value for the INTERVAL column</td>
<td>03:03:03</td>
</tr>
<tr>
<td>02 00 00 00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Welcome to the Analyzing Data Guide! This guide explains how to query and analyze data in your Vertica database.
**Queries**

Queries are database operations that retrieve data from one or more tables or views. In Vertica, the top-level SELECT statement is the query, and a query nested within another SQL statement is called a subquery.

Vertica is designed to run the same SQL standard queries that run on other databases. However, there are some differences between Vertica queries and queries used in other relational database management systems.

The Vertica **transaction model** is different from the SQL standard in a way that has a profound effect on query performance. You can:

- Run a query on a static backup of the database from any specific date and time. Doing so avoids holding locks or blocking other database operations.
- Use a subset of the standard SQL isolation levels and access modes (read/write or read-only) for a user session.

In Vertica, the primary structure of a SQL query is its statement. Each statement ends with a semicolon, and you can write multiple queries separated by semicolons; for example:

```sql
=> CREATE TABLE t1( ..., date_col date NOT NULL, ...);
=> CREATE TABLE t2( ..., state VARCHAR NOT NULL, ...);
```

**Multiple Instances of Dimension Tables in the FROM Clause**

The same dimension table can appear multiple times in a query's FROM clause, using different aliases. For example:

```sql
=> SELECT * FROM fact, dimension d1, dimension d2
WHERE fact.fk = d1.pk
AND
    fact.name = d2.name;
```
**Historical Queries**

Vertica can execute historical queries, which execute on a snapshot of the database taken at a specific timestamp or epoch. Historical queries return data only from the specified epoch. Because they do not return the latest data, historical queries hold no locks or blocking write operations.

Query results are private to the transaction and valid only for the length of the transaction. Query execution is the same regardless of the transaction isolation level.

You specify a historical query by qualifying the `SELECT` statement with one of the following clauses:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT EPOCH LATEST</td>
<td>Queries all data in the database up to but not including the current epoch. The result set includes data from the latest committed DML transaction</td>
</tr>
<tr>
<td>AT EPOCH <code>epoch-number</code></td>
<td>Queries all data in the database up to and including the specified epoch.</td>
</tr>
<tr>
<td>AT TIME <code>'timestamp'</code></td>
<td>Identifies the epoch that spans the specified timestamp.</td>
</tr>
</tbody>
</table>

**Restrictions**

- The specified epoch, or epoch of the specified timestamp, cannot be less than the Ancient History Mark epoch.

- Vertica silently ignores attempts to run historical queries on temporary tables.

---

Important: Any changes to a table schema are reflected across all epochs. For example, if you add a column to a table and specify a default value for it, all historical queries on that table display the new column and its default value.
Temporary Tables

You can use the CREATE TEMPORARY TABLE statement to implement certain queries using multiple steps:

1. Create one or more temporary tables.
2. Execute queries and store the result sets in the temporary tables.
3. Execute the main query using the temporary tables as if they were a normal part of the logical schema.

See CREATE TEMPORARY TABLE in the SQL Reference Manual for details.

SQL Queries

All DML (Data Manipulation Language) statements can contain queries. This section introduces some of the query types in Vertica, with additional details in later sections.

Note: Many of the examples in this chapter use the VMart schema. For information about other Vertica-supplied queries, see the Getting Started.

Simple Queries

Simple queries contain a query against one table. Minimal effort is required to process the following query, which looks for product keys and SKU numbers in the product table:

```sql
=> SELECT product_key, sku_number FROM public.product_dimension;

<table>
<thead>
<tr>
<th>product_key</th>
<th>sku_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>SKU-#129</td>
</tr>
<tr>
<td>87</td>
<td>SKU-#250</td>
</tr>
<tr>
<td>42</td>
<td>SKU-#125</td>
</tr>
<tr>
<td>49</td>
<td>SKU-#154</td>
</tr>
<tr>
<td>37</td>
<td>SKU-#107</td>
</tr>
<tr>
<td>36</td>
<td>SKU-#106</td>
</tr>
<tr>
<td>86</td>
<td>SKU-#248</td>
</tr>
<tr>
<td>41</td>
<td>SKU-#121</td>
</tr>
<tr>
<td>88</td>
<td>SKU-#257</td>
</tr>
<tr>
<td>40</td>
<td>SKU-#120</td>
</tr>
</tbody>
</table>

(10 rows)
```
Joins

Joins use a relational operator that combines information from two or more tables. The query's ON clause specifies how tables are combined, such as by matching foreign keys to primary keys. In the following example, the query requests the names of stores with transactions greater than 70 by joining the store key ID from the store schema's sales fact and sales tables:

```sql
=> SELECT store_name, COUNT(*) FROM store.store_sales_fact
JOIN store.store_dimension ON store.store_sales_fact.store_key = store.store_dimension.store_key
GROUP BY store_name HAVING COUNT(*) > 70 ORDER BY store_name;
```

<table>
<thead>
<tr>
<th>store_name</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store49</td>
<td>72</td>
</tr>
<tr>
<td>Store83</td>
<td>78</td>
</tr>
</tbody>
</table>

(2 rows)

For more detailed information, see Joins. See also the Multicolumn subqueries section in Subquery Examples.

Cross Joins

Also known as the Cartesian product, a cross join is the result of joining every record in one table with every record in another table. A cross join occurs when there is no join key between tables to restrict records. The following query, for example, returns all instances of vendor and store names in the vendor and store tables:

```sql
=> SELECT vendor_name, store_name FROM public.vendor_dimension
CROSS JOIN store.store_dimension;
```

<table>
<thead>
<tr>
<th>vendor_name</th>
<th>store_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deal Warehouse</td>
<td>Store41</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store12</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store46</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store50</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store15</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store48</td>
</tr>
<tr>
<td>Deal Warehouse</td>
<td>Store39</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store41</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store12</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store46</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store50</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store15</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store48</td>
</tr>
<tr>
<td>Sundry Wholesale</td>
<td>Store39</td>
</tr>
<tr>
<td>Market Discounters</td>
<td>Store41</td>
</tr>
<tr>
<td>Market Discounters</td>
<td>Store12</td>
</tr>
<tr>
<td>Market Discounters</td>
<td>Store46</td>
</tr>
</tbody>
</table>
This example's output is truncated because this particular cross join returned several thousand rows. See also Cross Joins.

Subqueries

A subquery is a query nested within another query. In the following example, we want a list of all products containing the highest fat content. The inner query (subquery) returns the product containing the highest fat content among all food products to the outer query block (containing query). The outer query then uses that information to return the names of the products containing the highest fat content.

```sql
=> SELECT product_description, fat_content FROM public.product_dimension WHERE fat_content IN (SELECT MAX(fat_content) FROM public.product_dimension WHERE category_description = 'Food' AND department_description = 'Bakery') LIMIT 10;
```

<table>
<thead>
<tr>
<th>product_description</th>
<th>fat_content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand #59110 hotdog buns</td>
<td>90</td>
</tr>
<tr>
<td>Brand #58107 english muffins</td>
<td>90</td>
</tr>
<tr>
<td>Brand #57135 english muffins</td>
<td>90</td>
</tr>
<tr>
<td>Brand #54870 cinnamon buns</td>
<td>90</td>
</tr>
<tr>
<td>Brand #53690 english muffins</td>
<td>90</td>
</tr>
<tr>
<td>Brand #53096 bagels</td>
<td>90</td>
</tr>
<tr>
<td>Brand #50678 chocolate chip cookies</td>
<td>90</td>
</tr>
<tr>
<td>Brand #49269 wheat bread</td>
<td>90</td>
</tr>
<tr>
<td>Brand #47156 coffee cake</td>
<td>90</td>
</tr>
<tr>
<td>Brand #43844 corn muffins</td>
<td>90</td>
</tr>
</tbody>
</table>

(10 rows)

For more information, see Subqueries.

Sorting Queries

Use the ORDER BY clause to order the rows that a query returns.
Special Note About Query Results

You could get different results running certain queries on one machine or another for the following reasons:

- **Partitioning** on a FLOAT type could return nondeterministic results because of the precision, especially when the numbers are close to one another, such as results from the RADIANS() function, which has a very small range of output.

  To get deterministic results, use NUMERIC if you must partition by data that is not an INTEGER type.

- Most analytics (with analytic aggregations, such as MIN()/MAX()/SUM()/COUNT()/AVG () as exceptions) rely on a unique order of input data to get deterministic result. If the analytic window-order clause cannot resolve ties in the data, results could be different each time you run the query.

  For example, in the following query, the analytic ORDER BY does not include the first column in the query, promotion_key. So for a tie of AVG(RADIANS(cost_dollar_amount)), product_version, the same promotion_key could have different positions within the analytic partition, resulting in a different NTILE() number. Thus, DISTINCT could also have a different result:

  ```sql
  => SELECT COUNT(*) FROM
      (SELECT DISTINCT SIN(FLOOR(MAX(store.store_sales_fact.promotion_key)))),
      NTILE(79) OVER(PARTITION BY AVG (RADIANS
      (store.store_sales_fact.cost_dollar_amount ))
      ORDER BY store.store_sales_fact.product_version)
FROM store.store_sales_fact
GROUP BY store.store_sales_fact.product_version,
       store.store_sales_fact.sales_dollar_amount ) AS store;
  
  count
  -------
  1425
  (1 row)
  
  If you add MAX(promotion_key) to analytic ORDER BY, the results are the same on any machine:
  ```
GROUP BY store.store_sales_fact.product_version,
    store.store_sales_fact.sales_dollar_amount) AS store;
Subqueries

A subquery is a SELECT statement embedded within another SELECT statement. The embedded subquery is often referenced as the query's inner statement, while the containing query is typically referenced as the query's statement, or outer query block. A subquery returns data that the outer query uses as a condition to determine what data to retrieve.

Like any query, a subquery returns records from a table that might consist of a single column and record, a single column with multiple records, or multiple columns and records. Queries can be noncorrelated or correlated. You can even use them to update or delete records in a table based on values that are stored in other database tables.

Note: Many examples in this section use the VMart database.
Subqueries Used in Search Conditions

Subqueries are used as search conditions in order to filter results. They specify the conditions for the rows returned from the containing query's select-list, a query expression, or the subquery itself. The operation evaluates to TRUE, FALSE, or UNKNOWN (NULL).

Syntax

```
search-condition {
    [ { AND | OR [ NOT ] } { predicate | ( search-condition ) } ]
} [ , ... ]
predicate
    { expression comparison-operator expression
    ... | string-expression [ NOT ] { LIKE | ILIKE | LIKEB | ILIKEB } string-expression
    ... | expression IS [ NOT ] NULL
    ... | expression [ NOT ] IN ( subquery | expression [ , ... n ] )
    ... | expression comparison-operator [ ANY | SOME ] ( subquery )
    ... | expression comparison-operator ALL ( subquery )
    ... | expression OR ( subquery )
    ... | [ NOT ] EXISTS ( subquery )
    ... | [ NOT ] IN ( subquery )
}
```

Parameters

<table>
<thead>
<tr>
<th>search-condition</th>
<th>Specifies the search conditions for the rows returned from one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Containing query's select-list</td>
</tr>
<tr>
<td></td>
<td>- Query expression</td>
</tr>
<tr>
<td></td>
<td>- Subquery</td>
</tr>
<tr>
<td></td>
<td>If the subquery is used with an UPDATE or DELETE statement, UPDATE specifies the rows to update and DELETE specifies the rows to delete.</td>
</tr>
</tbody>
</table>

| { AND | OR | NOT } | Keywords that specify the logical operators that combine conditions, or in the case of NOT, negate conditions. |
|-------------|---------------------------------------------------------------------------------------------------------------|
|             | - AND — Combines two conditions and evaluates                                                                |
| predicate | An expression that returns TRUE, FALSE, or UNKNOWN (NULL). |
| expression | A column name, constant, functiona scalar subquery, or combination of column names, constants, and functions connected by operators or subqueries. |
| comparison-operator | Test conditions between expressions, one of the following operators: |}
<p>| | • &lt; tests the condition of one expression being less than the other. |
| | • &gt; tests the condition of one expression being greater than the other. |
| | • &lt;= tests the condition of one expression being less than or equal to the other expression. |
| | • &gt;= tests the condition of one expression being greater than or equal to the other expression. |
| | • = tests the equality between two expressions. |
| | • &lt;=&gt; tests equality like the = operator, but it returns TRUE instead of UNKNOWN if both operands are UNKNOWN and FALSE instead of UNKNOWN if one operand is UNKNOWN. |
| | • &lt;&gt; and != test the condition of two expressions not equal to one another. |
| string-expression | A character string with optional wildcard (*) characters. |
| [ NOT ] { LIKE | ILIKE | LIKEB | ILIKEB } | Indicates that the character string following the |</p>
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS [ NOT ] NULL</td>
<td>Searches for values that are null or are not null.</td>
</tr>
<tr>
<td>ALL</td>
<td>Used with a comparison operator and a subquery. Returns TRUE for the lefthand predicate if all values returned by the subquery satisfy the comparison operation, or FALSE if not all values satisfy the comparison or if the subquery returns no rows to the outer query block.</td>
</tr>
<tr>
<td>ANY</td>
<td>SOME</td>
</tr>
<tr>
<td>[ NOT ] EXISTS</td>
<td>Used with a subquery to test for the existence of records that the subquery returns.</td>
</tr>
<tr>
<td>[ NOT ] IN</td>
<td>Searches for an expression on the basis of an expression's exclusion or inclusion from a list. The list of values is enclosed in parentheses and can be a subquery or a set of constants.</td>
</tr>
</tbody>
</table>

**Logical Operators AND and OR**

The AND and OR logical operators combine two conditions. AND evaluates to TRUE when both of the conditions joined by the AND keyword are matched, and OR evaluates to TRUE when either condition joined by OR is matched.

**OR Subqueries (complex expressions)**

Vertica supports subqueries in more complex expressions using OR; for example:
More than one subquery in the conjunct expression:

\[
(\text{SELECT MAX}(b) \text{ FROM } t1) + \text{SELECT} (\text{MAX FROM } t2) \text{ a IN (SELECT a FROM } t1) \text{ OR } b \text{ IN (SELECT } x \text{ FROM } t2)
\]

An OR clause in the conjunct expression involves at least one subquery:

\[
a \text{ IN (SELECT a FROM } t1) \text{ OR } b \text{ IN (SELECT } x \text{ FROM } t2) \text{ a IN (SELECT a FROM } t1) \text{ OR } b = 5
\]

One subquery is present but it is part of another expression:

\[
x \text{ IN (SELECT a FROM } t1) = (x = (\text{SELECT MAX FROM } t2) \text{ (x IN SELECT a FROM } t1) \text{ IS NULL}
\]

How AND Queries Are Evaluated

Vertica treats expressions separated by AND (conjunctive) operators individually. For example, if the WHERE clause were:

\[
\text{WHERE } (a \text{ IN (SELECT a FROM } t1) \text{ OR } b \text{ IN (SELECT } x \text{ FROM } t2)) \text{ AND } (c \text{ IN (SELECT a FROM } t1))
\]

the query would be interpreted as two conjunct expressions:

1. \((a \text{ IN (SELECT a FROM } t1) \text{ OR } b \text{ IN (SELECT } x \text{ FROM } t2))\)
2. \((c \text{ IN (SELECT a FROM } t1))\)

The first expression is considered a complex subquery, whereas the second expression is not.

Examples

The following list shows some of the ways you can filter complex conditions in the WHERE clause:

- **OR expression between a subquery and a non-subquery condition:**

  \[
  \Rightarrow \text{SELECT } x \text{ FROM } t \text{ WHERE } x > (\text{SELECT SUM(DISTINCT } x) \text{ FROM } t \text{ GROUP BY } y) \text{ OR } x < 9;
  \]

- **OR expression between two subqueries:**

  \[
  \Rightarrow \text{SELECT } * \text{ FROM } t \text{ WHERE } x=(\text{SELECT } x \text{ FROM } t) \text{ OR EXISTS(SELECT } x \text{ FROM } tt);
  \]
• Subquery expression:

```sql
=> SELECT * FROM t WHERE x=(SELECT x FROM t)+1 OR x<> (SELECT x FROM t)+1;
```

• OR expression with [NOT] IN subqueries:

```sql
=> SELECT * FROM t WHERE NOT (EXISTS (SELECT x FROM t)) OR x >9;
```

• OR expression with IS [NOT] NULL subqueries:

```sql
=> SELECT * FROM t WHERE (SELECT * FROM t) IS NULL OR (SELECT * FROM tt) IS NULL;
```

• OR expression with boolean column and subquery that returns Boolean data type:

```sql
=> SELECT * FROM t2 WHERE x = (SELECT x FROM t2) OR x;
```

**Note:** To return TRUE, the argument of OR must be a Boolean data type.

• OR expression in the CASE statement:

```sql
=> SELECT * FROM t WHERE CASE WHEN x=1 THEN x > (SELECT * FROM t)
  OR x < (SELECT * FROM t2) END;
```

• Analytic function, NULL-handling function, string function, math function, and so on:

```sql
=> SELECT x FROM t WHERE x > (SELECT COALESCE (x,y) FROM t GROUP BY x,y) OR
  x < 9;
```

• In user-defined functions (assuming \( f() \) is one):

```sql
=> SELECT * FROM t WHERE x > 5 OR x = (SELECT f(x) FROM t);
```

• Use of parentheses at different places to restructure the queries:

```sql
=> SELECT x FROM t WHERE (x = (SELECT x FROM t) AND y = (SELECT y FROM t))
  OR (SELECT x FROM t) =1;
```

• Multicolumn subqueries:

```sql
=> SELECT * FROM t WHERE (x,y) = (SELECT x,y FROM t) OR x > 5;
```
- Constant/NULL on lefthand side of subquery:

  => SELECT * FROM t WHERE x > 5 OR 5 = (SELECT x FROM t);

See Also

- Subquery Restrictions

In Place of an Expression

Subqueries that return a single value (unlike a list of values returned by IN subqueries) can be used just about anywhere an expression is allowed in SQL. It can be a column name, a constant, a function, a scalar subquery, or a combination of column names, constants, and functions connected by operators or subqueries.

For example:

  => SELECT c1 FROM t1 WHERE c1 = ANY (SELECT c1 FROM t2) ORDER BY c1;
  => SELECT c1 FROM t1 WHERE COALESCE((t1.c1 > ANY (SELECT c1 FROM t2)), TRUE);
  => SELECT c1 FROM t1 GROUP BY c1 HAVING COALESCE((t1.c1 <> ALL (SELECT c1 FROM t2)), TRUE);

Multi-column expressions are also supported:

  => SELECT c1 FROM t1 WHERE (t1.c1, t1.c2) = ALL (SELECT c1, c2 FROM t2);
  => SELECT c1 FROM t1 WHERE (t1.c1, t1.c2) <> ANY (SELECT c1, c2 FROM t2);

Vertica returns an error on queries where more than one row would be returned by any subquery used as an expression:

  => SELECT c1 FROM t1 WHERE c1 = (SELECT c1 FROM t2) ORDER BY c1;
  ERROR: more than one row returned by a subquery used as an expression

See Also

- Subquery Restrictions

Comparison Operators

Vertica supports Boolean subquery expressions in the WHERE clause with any of the following operators:
WHERE clause subqueries filter results and take the following form:

```
SELECT <column, ...> FROM <table>
WHERE <condition> (SELECT <column, ...> FROM <table> WHERE <condition>);
```

These conditions are available for all data types where comparison makes sense. All Comparison Operators are binary operators that return values of TRUE, FALSE, or UNKNOWN (NULL).

Expressions that correlate to just one outer table in the outer query block are supported, and these correlated expressions can be comparison operators.

The following subquery scenarios are supported:

```
SELECT * FROM T1 WHERE T1.x = (SELECT MAX(c1) FROM T2);
SELECT * FROM T1 WHERE T1.x >= (SELECT MAX(c1) FROM T2 WHERE T1.y = T2.c2);
SELECT * FROM T1 WHERE T1.x <= (SELECT MAX(c1) FROM T2 WHERE T1.y = T2.c2);
```

See Also

**Subquery Restrictions**

**LIKE Pattern Matching**

Vertica supports LIKE pattern-matching conditions in subqueries and take the following form:

```
string-expression [ NOT ] { LIKE | ILIKE | LIKEB | ILIKEB } string-expression
```

The following command searches for customers whose company name starts with "Ev" and returns the total count:

```
=> SELECT COUNT(*) FROM customer_dimension WHERE customer_name LIKE
    (SELECT 'Ev%' FROM customer_dimension LIMIT 1);

         count
----------
        153
(1 row)
```

Vertica also supports single-row subqueries as the pattern argument for LIKEB and ILIKEB predicates; for example:
=> SELECT * FROM t1 WHERE t1.x LIKEB (SELECT t2.x FROM t2);

The following symbols are substitutes for the LIKE keywords:

- ~~ LIKE
- ~# LIKEB
- ~~* ILIKE
- ~#* ILIKEB
- !~~ NOT LIKE
- !~# NOT LIKEB
- !~~* NOT ILIKE
- !~#* NOT IILIKEB

Note: The ESCAPE keyword is not valid for the above symbols.

See LIKE-predicate in the SQL Reference Manual for additional examples.

ANY and ALL

You typically use comparison operators (=, >, <, etc.) only on subqueries that return one row. With ANY and ALL operators, you can make comparisons on subqueries that return multiple rows.

These subqueries take the following form:

expression comparison-operator { ANY | ALL } (subquery)

ANY and ALL evaluate whether any or all of the values returned by a subquery match the left-hand expression.

Equivalent Operators

You can use following operators instead of ANY or ALL:

<table>
<thead>
<tr>
<th>This operator...</th>
<th>Is equivalent to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOME</td>
<td>ANY</td>
</tr>
<tr>
<td>IN</td>
<td>= ANY</td>
</tr>
<tr>
<td>NOT IN</td>
<td>&lt;&gt; ALL</td>
</tr>
</tbody>
</table>

Example Data

Examples below use the following tables and data:
ANY Subqueries

Subqueries that use the ANY keyword return true when any value retrieved in the subquery matches the value of the left-hand expression.

Examples

An ANY subquery within an expression:

```sql
=> SELECT c1, c2 FROM t1 WHERE COALESCE((t1.c1 > ANY (SELECT c1 FROM t2)));
c1 | c2    
---+-----
 2 | fed  
 2 | def  
 3 | ihg  
 3 | ghi  
 4 | jkl  
 5 | mno  
(6 rows)
```

ANY noncorrelated subqueries without aggregates:

```sql
=> SELECT c1 FROM t1 WHERE c1 = ANY (SELECT c1 FROM t2) ORDER BY c1;
c1    
----
 1    
 1    
 2    
 2    
 3    
 3    
(6 rows)
```

ANY noncorrelated subqueries with aggregates:

```sql
=> SELECT c1, c2 FROM t1 WHERE c1 <> ANY (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1 | c2   
----+-----
```
1 | cab
1 | abc
2 | fed
2 | def
4 | jkl
5 | mno
(6 rows)

=> SELECT c1 FROM t1 GROUP BY c1 HAVING c1 <> ANY (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1
----
1
2
4
5
(4 rows)

ANY noncorrelated subqueries with aggregates and a GROUP BY clause:

=> SELECT c1, c2 FROM t1 WHERE c1 <> ANY (SELECT MAX(c1) FROM t2 GROUP BY c2) ORDER BY c1;
c1 | c2
-----
1 | cab
1 | abc
2 | fed
2 | def
3 | ihg
3 | ghi
4 | jkl
5 | mno
(8 rows)

ANY noncorrelated subqueries with a GROUP BY clause:

=> SELECT c1, c2 FROM t1 WHERE c1 <> ANY (SELECT c1 FROM t2 GROUP BY c1) ORDER BY c1;
c1 | c2
-----
1 | cab
1 | abc
2 | fed
2 | def
3 | ihg
3 | ghi
(6 rows)

ANY correlated subqueries with no aggregates or GROUP BY clause:

=> SELECT c1, c2 FROM t1 WHERE c1 >= ANY (SELECT c1 FROM t2 WHERE t2.c2 = t1.c2) ORDER BY c1;
c1 | c2
-----
1 | abc
2 | fed
4 | jkl
(3 rows)
ALL Subqueries

A subquery that uses the **ALL** keyword returns true when all values retrieved by the subquery match the left-hand expression, otherwise it returns false.

Examples

ALL noncorrelated subqueries without aggregates:

```sql
=> SELECT c1, c2 FROM t1 WHERE c1 >= ALL (SELECT c1 FROM t2) ORDER BY c1;
c1 | c2
----+---
 3  | ihg
 3  | ghi
 4  | jkl
 5  | mno
(4 rows)
```

ALL noncorrelated subqueries with aggregates:

```sql
=> SELECT c1, c2 FROM t1 WHERE c1 = ALL (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1 | c2
----+---
 3  | ihg
 3  | ghi
(2 rows)
```

```sql
=> SELECT c1 FROM t1 GROUP BY c1 HAVING c1 <> ALL (SELECT MAX(c1) FROM t2) ORDER BY c1;
c1
----
 1
 2
 4
 5
(4 rows)
```

ALL noncorrelated subqueries with aggregates and a GROUP BY clause:

```sql
=> SELECT c1, c2 FROM t1 WHERE c1 <= ALL (SELECT MAX(c1) FROM t2 GROUP BY c2) ORDER BY c1;
c1 | c2
----+---
 1  | cab
 1  | abc
(2 rows)
```

ALL noncorrelated subqueries with a GROUP BY clause:

```sql
=> SELECT c1, c2 FROM t1 WHERE c1 <> ALL (SELECT c1 FROM t2 GROUP BY c1) ORDER BY c1;
c1 | c2
----+---
 4  | jkl
 5  | mno
(2 rows)
```
NULL Handling

Vertica supports multicolon < ALL subqueries where the columns are not marked NOT NULL. If any column contains a NULL value, Vertica returns a run-time error.

Vertica does not support = ANY subqueries that are nested within another expression if any column values are NULL.

See Also

Subquery Restrictions

EXISTS and NOT EXISTS

The EXISTS predicate is one of the most common predicates used to build conditions that use noncorrelated and correlated subqueries. Use EXISTS to identify the existence of a relationship without regard for the quantity. For example, EXISTS returns true if the subquery returns any rows, and [NOT] EXISTS returns true if the subquery returns no rows.

[NOT] EXISTS subqueries take the following form:

\[
\text{expression [ NOT ] EXISTS ( subquery )}
\]

The EXISTS condition is considered to be met if the subquery returns at least one row. Since the result depends only on whether any records are returned, and not on the contents of those records, the output list of the subquery is normally uninteresting. A common coding convention is to write all EXISTS tests as follows:

\[
\text{EXISTS (SELECT 1 WHERE ...)}
\]

In the above fragment, SELECT 1 returns the value 1 for every record in the query. If the query returns, for example, five records, it returns 5 ones. The system doesn't care about the real values in those records; it just wants to know if a row is returned.

Alternatively, a subquery's select list that uses EXISTS might consist of the asterisk (*). You do not need to specify column names, because the query tests for the existence or nonexistence of records that meet the conditions specified in the subquery.

\[
\text{EXISTS (SELECT * WHERE ...)}
\]

Notes

- If EXISTS (subquery) returns at least 1 row, the result is TRUE.
- If EXISTS (subquery) returns no rows, the result is FALSE.
If NOT EXISTS (subquery) returns at least 1 row, the result is FALSE.

If NOT EXISTS (subquery) returns no rows, the result is TRUE.

Examples

The following query retrieves the list of all the customers who purchased anything from any of the stores amounting to more than 550 dollars:

```sql
=> SELECT customer_key, customer_name, customer_state
    FROM public.customer_dimension
    WHERE EXISTS
        (SELECT 1 FROM store.store_sales_fact
         WHERE customer_key = public.customer_dimension.customer_key
         AND sales_dollar_amount > 550)
    AND customer_state = 'MA' ORDER BY customer_key;
```

<table>
<thead>
<tr>
<th>customer_key</th>
<th>customer_name</th>
<th>customer_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>14818</td>
<td>William X. Nielson</td>
<td>MA</td>
</tr>
<tr>
<td>18705</td>
<td>James J. Goldberg</td>
<td>MA</td>
</tr>
<tr>
<td>30231</td>
<td>Sarah N. McCabe</td>
<td>MA</td>
</tr>
<tr>
<td>48353</td>
<td>Mark L. Brown</td>
<td>MA</td>
</tr>
</tbody>
</table>

Whether you use EXISTS or IN subqueries depends on which predicates you select in outer and inner query blocks. For example, to get a list of all the orders placed by all stores on January 2, 2003 for vendors with records in the vendor table:

```sql
=> SELECT store_key, order_number, date_ordered
    FROM store.store_orders_fact
    WHERE EXISTS
        (SELECT 1 FROM public.vendor_dimension
         WHERE public.vendor_dimension.vendor_key = store.store_orders_fact.vendor_key)
    AND date_ordered = '2012-01-02';
```

<table>
<thead>
<tr>
<th>store_key</th>
<th>order_number</th>
<th>date_ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>2559</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>16</td>
<td>552</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>35</td>
<td>1156</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>13</td>
<td>3885</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>25</td>
<td>554</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>21</td>
<td>2687</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>49</td>
<td>3251</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>19</td>
<td>2922</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>26</td>
<td>1329</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>40</td>
<td>1183</td>
<td>2012-01-02</td>
</tr>
</tbody>
</table>

The above query looks for existence of the vendor and date ordered. To return a particular value, rather than simple existence, the query looks for orders placed by the vendor who got the best deal on January 4, 2004:

```sql
=> SELECT store_key, order_number, date_ordered
    FROM store.store_orders_fact ord, public.vendor_dimension vd
```
WHERE ord.vendor_key = vd.vendor_key AND vd.deal_size IN (SELECT MAX(deal_size) FROM public.vendor_dimension) AND date_ordered = '2013-01-04';

<table>
<thead>
<tr>
<th>store_key</th>
<th>order_number</th>
<th>date_ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>166</td>
<td>36008</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>113</td>
<td>66017</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>198</td>
<td>75716</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>27</td>
<td>150241</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>148</td>
<td>182207</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>9</td>
<td>188567</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>45</td>
<td>202416</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>24</td>
<td>250295</td>
<td>2013-01-04</td>
</tr>
<tr>
<td>121</td>
<td>251417</td>
<td>2013-01-04</td>
</tr>
</tbody>
</table>

(9 rows)

See Also

- Subquery Restrictions

IN and NOT IN

While you cannot equate a single value to a set of values, you can check to see if a single value is found within that set of values. Use the IN clause for multiple-record, single-column subqueries. After the subquery returns results introduced by IN or NOT IN, the outer query uses them to return the final result.

[NOT] IN subqueries take the following form:

```sql
{ expression [ NOT ] IN ( subquery ) | expression [ NOT ] IN ( expression ) }
```

There is no limit to the number of parameters passed to the IN clause of the SELECT statement; for example:

```sql
=> SELECT * FROM tablename WHERE column IN (a, b, c, d, e, ...);
```

Vertica also supports queries where two or more outer expressions refer to different inner expressions:

```sql
=> SELECT * FROM A WHERE (A.x,A.x) IN (SELECT B.x, B.y FROM B);
```

Examples

The following query uses the VMart schema to illustrate the use of outer expressions referring to different inner expressions:
To find all products supplied by stores in MA, first create the inner query and run it to ensure that it works as desired. The following query returns all stores located in MA:

```
=> SELECT store_key FROM store.store_dimension WHERE store_state = 'MA';
store_key
---------
13
31
(2 rows)
```

Then create the outer or main query that specifies all distinct products that were sold in stores located in MA. This statement combines the inner and outer queries using the IN predicate:

```
=> SELECT DISTINCT s.product_key, p.product_description
   FROM store.store_sales_fact s, public.product_dimension p
   WHERE s.product_key = p.product_key
     AND s.product_version = p.product_version
     AND s.store_key IN
       (SELECT store_key
        FROM store.store_dimension
        WHERE store_state = 'MA'))
   ORDER BY s.product_key;
product_key | product_description
-------------|---------------------
1 | Brand #1 white bread
1 | Brand #4 vegetable soup
3 | Brand #9 wheelchair
5 | Brand #15 cheddar cheese
5 | Brand #19 bleach
7 | Brand #22 canned green beans
7 | Brand #23 canned tomatoes
8 | Brand #24 champagne
8 | Brand #25 chicken nuggets
11 | Brand #32 sausage
... | ...
(281 rows)
```

When using NOT IN, the subquery returns a list of zero or more values in the outer query where the comparison column does not match any of the values returned from the subquery. Using the previous example, NOT IN returns all the products that are not supplied from MA.
Notes

Vertica supports multicolumn NOT IN subqueries in which the columns are not marked NOT NULL. If one of the columns is found to contain a NULL value during query execution, Vertica returns a run-time error.

Similarly, IN subqueries nested within another expression are not supported if any of the column values are NULL. For example, if in the following statement column x from either table contained a NULL value, Vertica returns a run-time error:

```sql
=> SELECT * FROM t1 WHERE (x IN (SELECT x FROM t2)) IS FALSE;
ERROR: NULL value found in a column used by a subquery
```

See Also

- Subquery Restrictions
- IN-predicate

Subqueries in the SELECT List

Subqueries can occur in the select list of the containing query. The results from the following statement are ordered by the first column (customer_name). You could also write ORDER BY 2 and specify that the results be ordered by the select-list subquery.

```sql
=> SELECT c.customer_name, (SELECT AVG(annual_income) FROM customer_dimension WHERE deal_size = c.deal_size) AVG_SAL_DEAL FROM customer_dimension c ORDER BY 1;
customer_name | AVG_SAL_DEAL
---------------+----------------
Goldstar       | 683429
Metatech       | 628086
Metadata       | 666728
Foodstar       | 695962
Verihope       | 715683
Veridata       | 868252
Bettercare     | 879156
Foodgen        | 958954
Virtacom       | 991551
Inicorp        | 1098835
...```
Notes

- Scalar subqueries in the select-list return a single row/column value. These subqueries use Boolean comparison operators: =, >, <, <=, >=.

  If the query is correlated, it returns NULL if the correlation results in 0 rows. If the query returns more than one row, the query errors out at run time and Vertica displays an error message that the scalar subquery must only return 1 row.

- Subquery expressions such as [NOT] IN, [NOT] EXISTS, ANY/SOME, or ALL always return a single Boolean value that evaluates to TRUE, FALSE, or UNKNOWN; the subquery itself can have many rows. Most of these queries can be correlated or noncorrelated.

  Note: ALL subqueries cannot be correlated.

- Subqueries in the ORDER BY and GROUP BY clauses are supported; for example, the following statement says to order by the first column, which is the select-list subquery:

  ```sql
g=> SELECT (SELECT MAX(x) FROM t2 WHERE y=t1.b) FROM t1 ORDER BY 1;
```

See Also

- Subquery Restrictions

WITH Clauses in SELECT

WITH clauses are concomitant queries within a larger, primary query. Vertica can evaluate WITH clauses in two ways:

- **Inline expansion** (default): Vertica evaluates each WITH clause every time it is referenced by the primary query.

- **Materialization**: Vertica evaluates each WITH clause once, stores results in a temporary table, and references this table as often as the query requires.

For details on WITH clause syntax and requirements, see **WITH Clause** in the SQL Reference Manual.
Inline Expansion of WITH Clause

By default, Vertica uses inline expansion to evaluate WITH clauses. Vertica evaluates each WITH clause every time it is referenced by the primary query. Inline expansion often works best if the query does not reference the same WITH clause multiple times, or if some local optimizations are possible after inline expansion.

Example

The following example shows a WITH clause that is a good candidate for inline expansion. The WITH clause is used in a query that obtains order information for all orders shipped in 2007, between December 15-31.

```sql
-- Disable the materialization method if previously enabled
ALTER SESSION SET PARAMETER EnableWithClauseMaterialization=1;

-- Begin WITH clause
WITH
    store_orders_fact_new AS
        (SELECT *
         FROM store.store_orders_fact
         WHERE date_shipped between '2016-12-15' and '2016-12-31')

-- End WITH clause
-- Begin main primary query
SELECT store_key, product_key, product_version, SUM(quantity_ordered*unit_price) AS total_price
FROM store_orders_fact_new
GROUP BY store_key, product_key, product_version
ORDER BY total_price;
```

Vertica processes the query as follows:

1. Expands the WITH clause reference to `store_orders_fact_new` within the primary query.
2. After expanding the WITH clause, evaluates the primary query.

Materialization of WITH Clause

When materialization is enabled, Vertica evaluates each WITH clause once, stores results in a temporary table, and references this table as often as the query requires. Vertica drops the temporary table after primary query execution completes.

Note: If the primary query returns with an error, temporary tables might be dropped only after the client’s session ends.
Materialization can facilitate better performance when WITH clauses are complex—for example, when the WITH clauses contain JOIN and GROUP BY clauses, and are referenced multiple times in the primary query.

If materialization is enabled, WITH statements perform an auto-commit of the user transaction. This occurs even when using EXPLAIN with the WITH statement.

Enabling Materialization

By default, materialization is disabled. You can enable and disable materialization by setting the configuration parameter ENABLE_WITH_CLAUSE_MATERIALIZATION at the following scopes:

- **Session**: Parameter setting remains in effect until you explicitly set or clear it, or the session ends.

  ```sql
  ALTER SESSION SET PARAMETER EnableWithClauseMaterialization={ 0 | 1 };
  ALTER SESSION CLEAR PARAMETER EnableWithClauseMaterialization;
  ```

- **Query**: WITH clause includes ENABLE_WITH_CLAUSE_MATERIALIZATION hint. Materialization is automatically cleared when the query returns.

  ```sql
  WITH /*+ENABLE_WITH_CLAUSE_MATERIALIZATION*/ with-query...
  ```

Example

The following example shows a WITH clause that is a good candidate for materialization. The query obtains data for the vendor who has the highest combined order cost for all orders:

```sql
-- Enable materialization
ALTER SESSION SET PARAMETER EnableWithClauseMaterialization=1;

-- Begin WITH clause, revenue
WITH
  revenue AS (SELECT vendor_key, SUM(total_order_cost) AS total_revenue
               FROM store.store_orders_fact
               GROUP BY vendor_key ORDER BY 1)

-- End defining WITH clause statement
-- Begin main primary query
SELECT vendor_name, vendor_address, vendor_city, total_revenue
FROM vendor_dimension v, revenue r
WHERE v.vendor_key = r.vendor_key AND total_revenue = (SELECT MAX(total_revenue) FROM revenue)
ORDER BY vendor_name;
```

Vertica processes this query as follows:
1. The WITH clause revenue evaluates its SELECT statement from table store.store_orders_fact.

2. The results of the revenue clause are stored in a local temporary table.

3. Whenever the revenue clause statement is referenced, the results stored in the table are used.

4. The temporary table is dropped when query execution is complete.

Noncorrelated and Correlated Subqueries

Subqueries can be categorized into two types:

- A noncorrelated (simple) subquery obtains its results independently of its containing (outer) statement.

- A correlated subquery requires values from its outer query in order to execute.

Noncorrelated Subqueries

A noncorrelated subquery executes independently of the outer query. The subquery executes first, and then passes its results to the outer query, For example:

```sql
=> SELECT name, street, city, state FROM addresses WHERE state IN (SELECT state FROM states);
```

Vertica executes this query as follows:

1. Executes the subquery SELECT state FROM states (in bold).

2. Passes the subquery results to the outer query.

A query's WHERE and HAVING clauses can specify noncorrelated subqueries if the subquery resolves to a single row, as shown below:

**In WHERE clause**

```sql
=> SELECT COUNT(*) FROM SubQ1 WHERE SubQ1.a = (SELECT y from SubQ2);
```

**In HAVING clause**

```sql
=> SELECT COUNT(*) FROM SubQ1 GROUP BY SubQ1.a HAVING SubQ1.a = (SubQ1.a & (SELECT y from SubQ2))
```
Correlated Subqueries

A correlated subquery typically obtains values from its outer query before it executes. When the subquery returns, it passes its results to the outer query.

**Note:** You can use an outer join to obtain the same effect as a correlated subquery.

In the following example, the subquery needs values from the `addresses.state` column in the outer query:

```sql
=> SELECT name, street, city, state
    FROM addresses
    WHERE EXISTS (SELECT * FROM states WHERE states.state = addresses.state);
```

Vertica executes this query as follows:

1. The subquery evaluates each `addresses.state` value in the outer block records.
2. It then passes its results to the outer query block.

Flattening FROM Clause Subqueries

**FROM clause** subqueries are always evaluated before their containing query. In some cases, the optimizer **flattens** FROM clause subqueries so the query can execute more efficiently.

For example, in order to create a query plan for the following statement, the Vertica query optimizer evaluates all records in table `t1` before it evaluates the records in table `t0`:

```sql
=> SELECT * FROM (SELECT a, MAX(a) AS max FROM (SELECT * FROM t1) AS t0 GROUP BY a);
```

Given the previous query, the optimizer can internally flatten it as follows:

```sql
=> SELECT * FROM (SELECT a, MAX(a) FROM t1 GROUP BY a) AS t0;
```

Both queries return the same results, but the flattened query runs more quickly.

Flattening Views

When a query's FROM clause specifies a **view**, the optimizer expands the view by replacing it with the query that the view encapsulates. If the view contains subqueries that are eligible for flattening, the optimizer produces a query plan that flattens those subqueries.
Flattening Restrictions

The optimizer cannot create a flattened query plan if a subquery or view contains one of the following elements:

- Aggregate function
- Analytic function
- Outer join (left, right or full)
- GROUP BY, ORDER BY, or HAVING clause
- DISTINCT keyword
- LIMIT or OFFSET clause
- UNION, EXCEPT, or INTERSECT clause
- EXISTS subquery

Examples

If a predicate applies to a view or subquery, the flattening operation can allow for optimizations by evaluating the predicates before the flattening takes place. Two examples follow.

View flattening

In this example, view v1 is defined as follows:

```
=> CREATE VIEW v1 AS SELECT * FROM a;
```

The following query specifies this view:

```
=> SELECT * FROM v1 JOIN b ON x=y WHERE x > 10;
```

Without flattening, the optimizer evaluates the query as follows:

1. Evaluates the subquery.
2. Applies the predicate WHERE x > 10.
In contrast, the optimizer can create a flattened query plan by applying the predicate before evaluating the subquery. This reduces the optimizer's work because it returns only the records WHERE \( x > 10 \) to the containing query.

Vertica internally transforms the previous query as follows:

```sql
=> SELECT * FROM (SELECT * FROM a) AS t1 JOIN b ON x=y WHERE x > 10;
```

The optimizer then flattens the query:

```sql
=> SELECT * FROM a JOIN b ON x=y WHERE x > 10;
```

**Subquery flattening**

The following example shows how Vertica transforms FROM clause subqueries within a WHERE clause IN subquery. Given the following query:

```sql
=> SELECT * FROM a WHERE b IN (SELECT b FROM (SELECT * FROM t2)) AS D WHERE x=1;
```

The optimizer flattens it as follows:

```sql
=> SELECT * FROM a WHERE b IN (SELECT b FROM t2) AS D WHERE x=1;
```

**See Also**

*Subquery Restrictions*

**Subqueries in UPDATE and DELETE Statements**

You can nest subqueries within UPDATE and DELETE statements.

**UPDATE Subqueries**

You can update records in one table according to values in others, by nesting a subquery within an UPDATE statement. The example below illustrates this through a couple of noncorrelated subqueries. You can reproduce this example with the following tables:

```sql
CREATE TABLE addresses(cust_id INTEGER, address VARCHAR(2000));
INSERT INTO addresses VALUES(20,'Lincoln Street');
INSERT INTO addresses VALUES(30,'Booth Hill Road');
INSERT INTO addresses VALUES(30,'Beach Avenue');
INSERT INTO addresses VALUES(40,'Mt. Vernon Street');
```
INSERT INTO addresses VALUES(50,'Hillside Avenue');

CREATE TABLE new_addresses(new_cust_id integer, new_address Boolean DEFAULT 'T');
INSERT INTO new_addresses VALUES(20);
INSERT INTO new_addresses VALUES(30);
INSERT INTO new_addresses VALUES(60,'F');
INSERT INTO new_addresses VALUES(80,'T');
COMMIT;

Queries on these tables return the following results:

=> SELECT * FROM addresses;
cust_id | address
--------|-------------------
      50 | Hillside Avenue
      30 | Booth Hill Road
      40 | Mt. Vernon Street
      20 | Lincoln Street
      30 | Beach Avenue
(5 rows)

=> SELECT * FROM new_addresses;
new_cust_id | new_address
-----------|-------------------
       30 | t
       20 | t
       80 | t
       60 | f
(4 rows)

1. The following UPDATE statement uses a noncorrelated subquery to join new_addresses and addresses records on customer IDs. UPDATE sets the value 'New Address' in the joined addresses records. The statement output indicates that three rows were updated:

=> UPDATE addresses SET address='New Address'
   WHERE cust_id IN (SELECT new_cust_id FROM new_addresses WHERE new_address='T');
OUTPUT
-------
 3
(1 row)

2. Query the addresses table to see the changes for matching customer ID 20 and 30. Addresses for customer ID 40 and 50 are not updated:

=> SELECT * FROM addresses;
cust_id | address
--------|-------------------
      20 | New Address
      30 | New Address
      30 | New Address
      40 | Mt. Vernon Street
      50 | Hillside Avenue
(5 rows)
DELETE Subqueries

You can delete records in one table based according to values in others by nesting a subquery within a DELETE statement.

For example, you want to remove records from new_addresses that were used earlier to update records in addresses. The following DELETE statement uses a noncorrelated subquery to join new_addresses and addresses records on customer IDs. It then deletes the joined records from table new_addresses:

```
=> DELETE FROM new_addresses
    WHERE new_cust_id IN (SELECT cust_id FROM addresses WHERE address='New Address');
OUTPUT
-------
  2
(1 row)
=> COMMIT;
COMMIT
```

Querying new_addresses confirms that the records were deleted:

```
=> SELECT * FROM new_addresses;
new_cust_id | new_address
-------------
   60 | f
   80 | t
(2 rows)
```

Subquery Examples

This topic illustrates some of the subqueries you can write. The examples use the VMart example database.

Single-Row Subqueries

Single-row subqueries are used with single-row comparison operators (=, >=, <=, <>, and <=>) and return exactly one row.

For example, the following query retrieves the name and hire date of the oldest employee in the Vmart database:
Vertica Documentation
Analyzing Data

```sql
=> SELECT employee_key, employee_first_name, employee_last_name, hire_date
   FROM employee_dimension
   WHERE hire_date = (SELECT MIN(hire_date) FROM employee_dimension);
```

<table>
<thead>
<tr>
<th>employee_first_name</th>
<th>employee_last_name</th>
<th>hire_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Bauer</td>
<td>1956-01-11</td>
</tr>
</tbody>
</table>

(1 row)

Multiple-Row Subqueries

Multiple-row subqueries return multiple records.

For example, the following IN clause subquery returns the names of the employees making the highest salary in each of the six regions:

```sql
=> SELECT employee_first_name, employee_last_name, annual_salary, employee_region
   FROM employee_dimension
   WHERE annual_salary IN
     (SELECT MAX(annual_salary) FROM employee_dimension GROUP BY employee_region)
   ORDER BY annual_salary DESC;
```

<table>
<thead>
<tr>
<th>employee_first_name</th>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>employee_region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexandra</td>
<td>Sanchez</td>
<td>992363</td>
<td>West</td>
</tr>
<tr>
<td>Mark</td>
<td>Vogel</td>
<td>983634</td>
<td>South</td>
</tr>
<tr>
<td>Tiffany</td>
<td>Vu</td>
<td>977716</td>
<td>SouthWest</td>
</tr>
<tr>
<td>Barbara</td>
<td>Lewis</td>
<td>957949</td>
<td>Midwest</td>
</tr>
<tr>
<td>Sally</td>
<td>Gauthier</td>
<td>927335</td>
<td>East</td>
</tr>
<tr>
<td>Wendy</td>
<td>Nielson</td>
<td>777037</td>
<td>NorthWest</td>
</tr>
</tbody>
</table>

(6 rows)

Multicolumn Subqueries

Multicolumn subqueries return one or more columns. Sometimes a subquery's result set is evaluated in the containing query in column-to-column and row-to-row comparisons.

**Note:** Multicolumn subqueries can use the <>, !=, and = operators but not the <, >, <=, >= operators.

You can substitute some multicolumn subqueries with a join, with the reverse being true as well. For example, the following two queries ask for the sales transactions of all products sold online to customers located in Massachusetts and return the same result set. The only difference is the first query is written as a join and the second is written as a subquery.

**Join query:**

```sql
=> SELECT *
   FROM online_sales.online_sales_fact
   INNER JOIN public.customer_dimension
   USING (customer_key);
```

**Subquery:**

```sql
=> SELECT *
   FROM online_sales.online_sales_fact
   WHERE customer_key IN
     (SELECT customer_key
      FROM customer_dimension
      WHERE region = 'Massachusetts');
```
The following query returns all employees in each region whose salary is above the average:

```sql
=> SELECT e.employee_first_name, e.employee_last_name, e.annual_salary,
    e.employee_region, s.average
FROM employee_dimension e,
    (SELECT employee_region, AVG(annual_salary) AS average
     FROM employee_dimension GROUP BY employee_region) AS s
WHERE e.employee_region = s.employee_region AND e.annual_salary > s.average
ORDER BY annual_salary DESC;
```

<table>
<thead>
<tr>
<th>employee_first_name</th>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>employee_region</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doug</td>
<td>Overstreet</td>
<td>995533</td>
<td>East</td>
<td>61192.786013986</td>
</tr>
<tr>
<td>Matt</td>
<td>Gauthier</td>
<td>988807</td>
<td>South</td>
<td>57337.8638902996</td>
</tr>
<tr>
<td>Lauren</td>
<td>Nguyen</td>
<td>968625</td>
<td>West</td>
<td>56848.4274914089</td>
</tr>
<tr>
<td>Jack</td>
<td>Campbell</td>
<td>963914</td>
<td>West</td>
<td>56848.4274914089</td>
</tr>
<tr>
<td>William</td>
<td>Martin</td>
<td>943477</td>
<td>NorthWest</td>
<td>58928.2276119403</td>
</tr>
<tr>
<td>Luigi</td>
<td>Campbell</td>
<td>939255</td>
<td>Midwest</td>
<td>59614.9170454545</td>
</tr>
<tr>
<td>Sarah</td>
<td>Brown</td>
<td>901619</td>
<td>South</td>
<td>57337.8638902996</td>
</tr>
<tr>
<td>Craig</td>
<td>Goldberg</td>
<td>895836</td>
<td>East</td>
<td>61192.786013986</td>
</tr>
<tr>
<td>Sam</td>
<td>Vu</td>
<td>889841</td>
<td>Midwest</td>
<td>59614.9170454545</td>
</tr>
<tr>
<td>Luigi</td>
<td>Sanchez</td>
<td>885078</td>
<td>Midwest</td>
<td>59614.9170454545</td>
</tr>
<tr>
<td>Michael</td>
<td>Weaver</td>
<td>882685</td>
<td>South</td>
<td>57337.8638902996</td>
</tr>
<tr>
<td>Doug</td>
<td>Pavlov</td>
<td>881443</td>
<td>SouthWest</td>
<td>57187.2510548523</td>
</tr>
<tr>
<td>Ruth</td>
<td>McNulty</td>
<td>874897</td>
<td>East</td>
<td>61192.786013986</td>
</tr>
<tr>
<td>Luigi</td>
<td>Dobisz</td>
<td>868213</td>
<td>West</td>
<td>56848.4274914089</td>
</tr>
<tr>
<td>Laura</td>
<td>Lang</td>
<td>865829</td>
<td>East</td>
<td>61192.786013986</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You can also use the **EXCEPT, INTERSECT, and UNION [ALL]** keywords in FROM, WHERE, and HAVING clauses.

The following subquery returns information about all Connecticut-based customers who bought items through either stores or online sales channel and whose purchases amounted to more than 500 dollars:

```sql
=> SELECT DISTINCT customer_key, customer_name FROM public.customer_dimension
WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact
WHERE sales_dollar_amount > 500
UNION ALL
SELECT customer_key FROM online_sales.online_sales_fact
WHERE sales_dollar_amount > 500)
AND customer_state = 'CT';
```

<table>
<thead>
<tr>
<th>customer_key</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Carla Y. Kramer</td>
</tr>
<tr>
<td>733</td>
<td>Mary Z. Vogel</td>
</tr>
<tr>
<td>931</td>
<td>Lauren X. Roy</td>
</tr>
<tr>
<td>1533</td>
<td>James C. Vu</td>
</tr>
<tr>
<td>2948</td>
<td>Infocare</td>
</tr>
<tr>
<td>4989</td>
<td>Matt Z. Winkler</td>
</tr>
<tr>
<td>5311</td>
<td>John Z. Goldberg</td>
</tr>
<tr>
<td>5520</td>
<td>Laura M. Martin</td>
</tr>
<tr>
<td>5623</td>
<td>Daniel R. Kramer</td>
</tr>
</tbody>
</table>
HAVING Clause Subqueries

A HAVING clause is used in conjunction with the GROUP BY clause to filter the select-list records that a GROUP BY returns. HAVING clause subqueries must use Boolean comparison operators: =, >, <, <=, >= and take the following form:

```sql
SELECT <column, ...>
FROM <table>
GROUP BY <expression>
HAVING <expression>
(SELECT <column, ...>
FROM <table>
HAVING <expression>);
```

For example, the following statement uses the VMart database and returns the number of customers who purchased lowfat products. Note that the GROUP BY clause is required because the query uses an aggregate (COUNT).

```sql
=> SELECT s.product_key, COUNT(s.customer_key) FROM store.store_sales_fact s
  GROUP BY s.product_key HAVING s.product_key IN
  (SELECT product_key FROM product_dimension WHERE diet_type = 'Low Fat');
```

The subquery first returns the product keys for all low-fat products, and the outer query then counts the total number of customers who purchased those products.

<table>
<thead>
<tr>
<th>product_key</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
</tr>
<tr>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>106</td>
<td>1</td>
</tr>
<tr>
<td>118</td>
<td>1</td>
</tr>
<tr>
<td>169</td>
<td>1</td>
</tr>
<tr>
<td>181</td>
<td>2</td>
</tr>
<tr>
<td>184</td>
<td>2</td>
</tr>
<tr>
<td>186</td>
<td>2</td>
</tr>
<tr>
<td>211</td>
<td>1</td>
</tr>
<tr>
<td>229</td>
<td>1</td>
</tr>
<tr>
<td>267</td>
<td>1</td>
</tr>
<tr>
<td>289</td>
<td>1</td>
</tr>
<tr>
<td>334</td>
<td>2</td>
</tr>
<tr>
<td>336</td>
<td>1</td>
</tr>
</tbody>
</table>

(15 rows)
Subquery Restrictions

The following list summarizes subquery restrictions in Vertica.

- Subqueries are not allowed in the defining query of a `CREATE PROJECTION` statement.

- Subqueries can be used in the `SELECT` list, but `GROUP BY` or aggregate functions are not allowed in the query if the subquery is not part of the `GROUP BY` clause in the containing query. For example, the following two statement returns an error message:

  ```
  => SELECT y, (SELECT MAX(a) FROM t1) FROM t2 GROUP BY y;
  ERROR: subqueries in the SELECT or ORDER BY are not supported if the subquery is not part of the GROUP BY
  => SELECT MAX(y), (SELECT MAX(a) FROM t1) FROM t2;
  ERROR: subqueries in the SELECT or ORDER BY are not supported if the query has aggregates and the subquery is not part of the GROUP BY
  ```

- Subqueries are supported within `UPDATE` statements with the following exceptions:
  - You cannot use `SET column = {expression}` to specify a subquery.
  - The table specified in the `UPDATE` list cannot also appear in the `FROM` list (no self joins).
  - `FROM` clause subqueries require an alias but tables do not. If the table has no alias, the query must refer to columns inside it as `table-name.column-name`. However, if column names are uniquely identified among all tables used by the query, then column names do not need to include their table name.
  - If the `ORDER BY` clause is inside a `FROM` clause subquery, rather than in the containing query, the query could return unexpected sort results. This is because Vertica data comes from multiple nodes, so sort order cannot be guaranteed unless an `ORDER BY` clause is specified in the outer query block. This behavior is compliant with the SQL standard but it might differ from other databases.

- Multicolumn subqueries cannot use the `<`, `>`, `<=`, `>=` comparison operators. They can use `<>`, `! =`, and `=` operators.

- `WHERE` and `HAVING` clause subqueries must use Boolean comparison operators: `=`, `>`, `<`, `<>`, `<=`, `>=`. Those subqueries can be noncorrelated and correlated.

- `[NOT] IN` and `ANY` subqueries nested within another expression are not supported if any of the column values are NULL. In the following statement, for example, if column x from either table t1 or t2 contains a NULL value, Vertica returns a run-time error:
Vertica returns an error message during subquery run time on scalar subqueries that return more than one row.

Aggregates and GROUP BY clauses are allowed in subqueries, as long as those subqueries are not correlated.

Correlated expressions under ALL and [NOT] IN are not supported.

Correlated expressions under OR are not supported.

Multiple correlations are allowed only for subqueries that are joined with an equality predicate (<, >, <=, >=, =, <>), but IN/NOT IN, EXISTS/NOT EXISTS predicates within correlated subqueries are not allowed:

```
=> SELECT t2.x, t2.y, t2.z FROM t2 WHERE t2.z NOT IN (SELECT t1.z FROM t1 WHERE t1.x = t2.x);
ERROR: Correlated subquery with NOT IN is not supported
```

Up to one level of correlated subqueries is allowed in the WHERE clause if the subquery references columns in the immediate outer query block. For example, the following query is not supported because the `t2.x = t3.x` subquery can only refer to table `t1` in the outer query, making it a correlated expression because `t3.x` is two levels out:

```
=> SELECT t3.x, t3.y, t3.z FROM t3 WHERE t3.z IN (SELECT t1.z FROM t1 WHERE EXISTS (SELECT 'x' FROM t2 WHERE t2.x = t3.x) AND t1.x = t3.x);
ERROR: More than one level correlated subqueries are not supported
```

The query is supported if it is rewritten as follows:

```
=> SELECT t3.x, t3.y, t3.z FROM t3 WHERE t3.z IN (SELECT t1.z FROM t1 WHERE EXISTS (SELECT 'x' FROM t2 WHERE t2.x = t1.x) AND t1.x = t3.x);
```
Joins

Queries can combine records from multiple tables, or multiple instances of the same table. A query that combines records from one or more tables is called a join. Joins are allowed in SELECT statements and subqueries.

Supported Join Types

Vertica supports the following join types:

- Inner (including natural, cross) joins
- Left, right, and full outer joins
- Optimizations for equality and range joins predicates

Vertica does not support nested loop joins.

Join Algorithms

Vertica’s query optimizer implements joins with either the hash join or merge join algorithm. For details, see Hash Joins Versus Merge Joins.

Join Syntax

Vertica supports the ANSI SQL-92 standard for joining tables, as follows:

```
(table-reference [join-type] JOIN table-reference [ ON join-predicate ])
```

where `join-type` can be one of the following:

- **INNER** (default)
- **LEFT** [ OUTER ]
- **RIGHT** [ OUTER ]
- **FULL** [ OUTER ]
• **NATURAL**

• **CROSS**

For example:

```sql
=> SELECT * FROM T1 INNER JOIN T2 ON T1.id = T2.id;
```

**Note:** The ON *join-predicate* clause is invalid for NATURAL and CROSS joins, required for all other join types.

### Alternative Syntax Options

Vertica also supports two older join syntax conventions:

**Join specified by WHERE clause join predicate**

INNER JOIN is equivalent to a query that specifies its join predicate in a WHERE clause. For example, this example and the previous one return equivalent results. They both specify an inner join between tables T1 and T2 on columns T1.id and T2.id, respectively.

```sql
=> SELECT * FROM T1, T2 WHERE T1.id = T2.id;
```

**Join specified by USING clause**

You can join two tables on identically named columns in a USING clause. For example:

```sql
=> SELECT * FROM T1 INNER JOIN T2 USING(id);
```

The INNER keyword is optional; a join that is specified by a USING clause is always an inner join. Thus, this example and the previous ones return equivalent results.

### Benefits of SQL-92 Join Syntax

Vertica recommends that you use SQL-92 join syntax for several reasons:

- SQL-92 outer join syntax is portable across databases; the older syntax was not consistent between databases.

- SQL-92 syntax provides greater control over whether predicates are evaluated during or after outer joins. This was also not consistent between databases when using the older syntax.
SQL-92 syntax eliminates ambiguity in the order of evaluating the joins, in cases where more than two tables are joined with outer joins.

Join Conditions vs. Filter Conditions

If you do not use the SQL-92 syntax, join conditions (predicates that are evaluated during the join) are difficult to distinguish from filter conditions (predicates that are evaluated after the join), and in some cases cannot be expressed at all. With SQL-92, join conditions and filter conditions are separated into two different clauses, the ON clause and the WHERE clause, respectively, making queries easier to understand.

- The ON clause contains relational operators (for example, <, <=, >, >=, <>, =, <=>) or other predicates that specify which records from the left and right input relations to combine, such as by matching foreign keys to primary keys. ON can be used with inner, left outer, right outer, and full outer joins. Cross joins and union joins do not use an ON clause.

  Inner joins return all pairings of rows from the left and right relations for which the ON clause evaluates to TRUE. In a left join, all rows from the left relation in the join are present in the result; any row of the left relation that does not match any rows in the right relation is still present in the result but with nulls in any columns taken from the right relation. Similarly, a right join preserves all rows from the right relation, and a full join retains all rows from both relations.

- The WHERE clause is evaluated after the join is performed. It filters records returned by the FROM clause, eliminating any records that do not satisfy the WHERE clause condition.

Vertica automatically converts outer joins to inner joins in cases where it is correct to do so, allowing the optimizer to choose among a wider set of query plans and leading to better performance.
**Inner Joins**

An inner join combines records from two tables based on a join predicate and requires that each record in the first table has a matching record in the second table. Thus, inner joins return only records from both joined tables that satisfy the join condition. Records that contain no matches are excluded from the result set.

Inner joins take the following form:

```sql
SELECT column-list FROM left-join-table
[INNER] JOIN right-join-table ON join-predicate
```

If you omit the INNER keyword, Vertica assumes an inner join. Inner joins are commutative and associative. You can specify tables in any order without changing the results.

**Example**

The following example specifies an inner join between tables `store.store_dimension` and `public.employee_dimension` whose records have matching values in columns `store_region` and `employee_region`, respectively:

```sql
=> SELECT s.store_region, SUM(e.vacation_days) TotalVacationDays
    FROM public.employee_dimension e,
    store.store_dimension s
    WHERE s.store_region=e.employee_region
    GROUP BY s.store_region ORDER BY TotalVacationDays;
```

This join can also be expressed as follows:

```sql
=> SELECT s.store_region, SUM(e.vacation_days) TotalVacationDays
    FROM public.employee_dimension e, store.store_dimension s
    WHERE s.store_region=e.employee_region
    GROUP BY s.store_region ORDER BY TotalVacationDays;
```

Both queries return the same result set:

<table>
<thead>
<tr>
<th>store_region</th>
<th>TotalVacationDays</th>
</tr>
</thead>
<tbody>
<tr>
<td>NorthWest</td>
<td>23280</td>
</tr>
<tr>
<td>Southwest</td>
<td>367250</td>
</tr>
<tr>
<td>Midwest</td>
<td>925938</td>
</tr>
<tr>
<td>South</td>
<td>1280468</td>
</tr>
<tr>
<td>East</td>
<td>1952854</td>
</tr>
<tr>
<td>West</td>
<td>2849976</td>
</tr>
</tbody>
</table>

(6 rows)
If the join's inner table store.store_dimension has any rows with store_region values that do not match employee_region values in table public.employee_dimension, those rows are excluded from the result set. To include that row, you can specify an outer join.

Equi-Joins and Non Equi-Joins

Vertica supports any arbitrary join expression with both matching and non-matching column values. For example:

```sql
SELECT * FROM Fact JOIN Dim ON fact.x = dim.x;
SELECT * FROM Fact JOIN Dim ON fact.x > dim.y;
SELECT * FROM Fact JOIN Dim ON fact.x <= dim.y;
SELECT * FROM Fact JOIN Dim ON fact.x <> dim.y;
SELECT * FROM Fact JOIN Dim ON fact.x <= dim.y;
```

**Note:** Operators = and <=> generally run the fastest.

Equi-joins are based on equality (matching column values). This equality is indicated with an equal sign (=), which functions as the comparison operator in the ON clause using SQL-92 syntax or the WHERE clause using older join syntax.

The first example below uses SQL-92 syntax and the ON clause to join the online sales table with the call center table using the call center key; the query then returns the sale date key that equals the value 156:

```sql
=> SELECT sale_date_key, cc_open_date FROM online_sales.online_sales_fact
   INNER JOIN online_sales.call_center_dimension
   ON online_sales.online_sales_fact.call_center_key =
   online_sales.call_center_dimension.call_center_key
   AND sale_date_key = 156);
   sale_date_key | cc_open_date
   ---------------+---------------
   156 | 2005-08-12
(1 row)
```

The second example uses older join syntax and the WHERE clause to join the same tables to get the same results:

```sql
=> SELECT sale_date_key, cc_open_date
   FROM online_sales.online_sales_fact, online_sales.call_center_dimension
   WHERE online_sales.online_sales_fact.call_center_key =
   online_sales.call_center_dimension.call_center_key
   AND sale_date_key = 156;
   sale_date_key | cc_open_date
   ---------------+---------------
   156 | 2005-08-12
(1 row)
```

Vertica also permits tables with compound (multi-column) primary and foreign keys. For example, to create a pair of tables with multi-column keys:
CREATE TABLE dimension(pk1 INTEGER NOT NULL, pk2 INTEGER NOT NULL);
ALTER TABLE dimension ADD PRIMARY KEY (pk1, pk2);
CREATE TABLE fact (fk1 INTEGER NOT NULL, fk2 INTEGER NOT NULL);
ALTER TABLE fact ADD FOREIGN KEY (fk1, fk2) REFERENCES dimension (pk1, pk2);

To join tables using compound keys, you must connect two join predicates with a Boolean AND operator. For example:

SELECT * FROM fact f JOIN dimension d ON f.fk1 = d.pk1 AND f.fk2 = d.pk2;

You can write queries with expressions that contain the <=> operator for NULL=NULL joins.

SELECT * FROM fact JOIN dim ON fact.x <=> dim.y;

The <=> operator performs an equality comparison like the = operator, but it returns true, instead of NULL, if both operands are NULL, and false, instead of NULL, if one operand is NULL.

SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
----------|-----------|-----------
t | t | f
(1 row)

Compare the <=> operator to the = operator:

SELECT 1 = 1, NULL = NULL, 1 = NULL;
----------|-----------|-----------
t | | |
(1 row)

Note: Writing NULL=NULL joins on primary key/foreign key combinations is not an optimal choice because PK/FK columns are usually defined as NOT NULL.

When composing joins, it helps to know in advance which columns contain null values. An employee's hire date, for example, would not be a good choice because it is unlikely hire date would be omitted. An hourly rate column, however, might work if some employees are paid hourly and some are salaried. If you are unsure about the value of columns in a given table and want to check, type the command:

SELECT COUNT(*) FROM tablename WHERE columnname IS NULL;

Natural Joins

A natural join is just a join with an implicit join predicate. Natural joins can be inner, left outer, right outer, or full outer joins and take the following form:
Natural joins are, by default, natural inner joins; however, there can also be natural (left/right) outer joins. The primary difference between an inner and natural join is that inner joins have an explicit join condition, whereas the natural join’s conditions are formed by matching all pairs of columns in the tables that have the same name and compatible data types, making natural joins equi-joins because join condition are equal between common columns. (If the data types are incompatible, Vertica returns an error.)

**Note:** The Data Type Coercion Chart lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

The following query is a simple natural join between tables T1 and T2 when the T2 column `val` is greater than 5:

```sql
=> SELECT * FROM T1 NATURAL JOIN T2 WHERE T2.val > 5;
```

The following example shows a natural join between the `store_sales_fact` table and the `product_dimension` table with columns of the same name, `product_key` and `product_version`:

```sql
=> SELECT product_description, store.store_sales_fact.*
    FROM store.store_sales_fact, public.product_dimension
    WHERE store.store_sales_fact.product_key = public.product_dimension.product_key
    AND store.store_sales_fact.product_version = public.product_dimension.product_version;
```

The following three queries return the same result expressed as a basic query, an inner join, and a natural join. at the table expressions are equivalent only if the common attribute in the `store_sales_fact` table and the `store_dimension` table is `store_key`. If both tables have a column named `store_key`, then the natural join would also have a `store_sales_fact.store_key = store_dimension.store_key` join condition. Since the results are the same in all three instances, they are shown in the first (basic) query only:

```sql
=> SELECT store_name FROM store.store_sales_fact, store.store_dimension
    WHERE store.store_sales_fact.store_key = store.store_dimension.store_key
    AND store.store_dimension.store_state = 'MA' ORDER BY store_name;

store_name
-----------
Store11
Store128
Store178
Store66
Store8
Store90
(6 rows)
```

The query written as an inner join:
In the case of the natural join, the join predicate appears implicitly by comparing all of the columns in both tables that are joined by the same column name. The result set contains only one column representing the pair of equally-named columns.

```
=> SELECT store_name FROM store.store_sales_fact
    NATURAL JOIN store.store_dimension
    WHERE store.store_dimension.store_state = 'MA' ORDER BY store_name;
```

Cross Joins

Cross joins are the simplest joins to write, but they are not usually the fastest to run because they consist of all possible combinations of two tables’ records. Cross joins contain no join condition and return what is known as a Cartesian product, where the number of rows in the result set is equal to the number of rows in the first table multiplied by the number of rows in the second table.

The following query returns all possible combinations from the promotion table and the store sales table:

```
=> SELECT * FROM promotion_dimension CROSS JOIN store.store_sales_fact;
```

Because this example returns over 600 million records, many cross join results can be extremely large and difficult to manage. Cross joins can be useful, however, such as when you want to return a single-row result set.

**Tip:** Filter out unwanted records in a cross with WHERE clause join predicates:

```
=> SELECT * FROM promotion_dimension p CROSS JOIN store.store_sales_fact f
    WHERE p.promotion_key LIKE f.promotion_key;
```

Implicit versus Explicit Joins

Vertica recommends that you do not write implicit cross joins (comma-separated tables in the FROM clause). These queries can imply accidental omission of a join predicate.

The following query implicitly cross joins tables promotion_dimension and store.store_sales_fact:

```
=> SELECT * FROM promotion_dimension, store.store_sales_fact;
```

It is better practice to express this cross join explicitly, as follows:
Examples

The following example creates two small tables and their superprojections and then runs a cross join on the tables:

```sql
=> CREATE TABLE employee(employee_id INT, employee_fname VARCHAR(50));
=> CREATE TABLE department(dept_id INT, dept_name VARCHAR(50));
=> INSERT INTO employee VALUES (1, 'Andrew');
=> INSERT INTO employee VALUES (2, 'Priya');
=> INSERT INTO employee VALUES (3, 'Michelle');
=> INSERT INTO department VALUES (1, 'Engineering');
=> INSERT INTO department VALUES (2, 'QA');
=> SELECT * FROM employee CROSS JOIN department;
```

In the result set, the cross join retrieves records from the first table and then creates a new row for every row in the 2nd table. It then does the same for the next record in the first table, and so on.

```
<table>
<thead>
<tr>
<th>employee_id</th>
<th>employee_name</th>
<th>dept_id</th>
<th>dept_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Andrew</td>
<td>1</td>
<td>Engineering</td>
</tr>
<tr>
<td>2</td>
<td>Priya</td>
<td>1</td>
<td>Engineering</td>
</tr>
<tr>
<td>3</td>
<td>Michelle</td>
<td>1</td>
<td>Engineering</td>
</tr>
<tr>
<td>1</td>
<td>Andrew</td>
<td>2</td>
<td>QA</td>
</tr>
<tr>
<td>2</td>
<td>Priya</td>
<td>2</td>
<td>QA</td>
</tr>
<tr>
<td>3</td>
<td>Michelle</td>
<td>2</td>
<td>QA</td>
</tr>
</tbody>
</table>
(6 rows)
```

Outer Joins

Outer joins extend the functionality of inner joins by letting you preserve rows of one or both tables that do not have matching rows in the non-preserved table. Outer joins take the following form:

```sql
SELECT column_list FROM left-join-table
[ LEFT | RIGHT | FULL ] OUTER JOIN right-join-table ON join-predicate
```

Note: Omitting the keyword OUTER from your statements does not affect results of left and right joins. LEFT OUTER JOIN and LEFT JOIN perform the same operation and return the same results.
Left Outer Joins

A left outer join returns a complete set of records from the left-joined (preserved) table T1, with matched records, where available, in the right-joined (non-preserved) table T2. Where Vertica finds no match, it extends the right side column (T2) with null values.

=> SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.x = T2.x;

To exclude the non-matched values from T2, write the same left outer join, but filter out the records you don’t want from the right side by using a WHERE clause:

=> SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.x = T2.x WHERE T2.x IS NOT NULL;

The following example uses a left outer join to enrich telephone call detail records with an incomplete numbers dimension. It then filters out results that are known not to be from Massachusetts:

=> SELECT COUNT(*) FROM calls LEFT OUTER JOIN numbers ON calls.to_phone = numbers.phone WHERE NVL(numbers.state, '') <> 'MA';

Right Outer Joins

A right outer join returns a complete set of records from the right-joined (preserved) table, as well as matched values from the left-joined (non-preserved) table. If Vertica finds no matching records from the left-joined table (T1), NULL values appears in the T1 column for any records with no matching values in T1. A right join is, therefore, similar to a left join, except that the treatment of the tables is reversed.

=> SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.x = T2.x;

The above query is equivalent to the following query, where T1 RIGHT OUTER JOIN T2 = T2 LEFT OUTER JOIN T1.

=> SELECT * FROM T2 LEFT OUTER JOIN T1 ON T2.x = T1.x;

The following example identifies customers who have not placed an order:

=> SELECT customers.customer_id FROM orders RIGHT OUTER JOIN customers ON orders.customer_id = customers.customer_id GROUP BY customers.customer_id HAVING COUNT(orders.customer_id) = 0;
**Full Outer Joins**

A full outer join returns results for both left and right outer joins. The joined table contains all records from both tables, including nulls (missing matches) from either side of the join. This is useful if you want to see, for example, each employee who is assigned to a particular department and each department that has an employee, but you also want to see all the employees who are not assigned to a particular department, as well as any department that has no employees:

```sql
=> SELECT employee_last_name, hire_date FROM employee_dimension emp
    FULL OUTER JOIN department dept ON emp.employee_key = dept.department_key;
```

**Notes**

Vertica also supports joins where the outer (preserved) table or subquery is replicated on more than one node and the inner (non-preserved) table or subquery is segmented across more than one node. For example, in the following query, the fact table, which is almost always segmented, appears on the non-preserved side of the join, and it is allowed:

```sql
=> SELECT sales_dollar_amount, transaction_type, customer_name
      FROM store.store_sales_fact f RIGHT JOIN customer_dimension d
          ON f.customer_key = d.customer_key;
```

<table>
<thead>
<tr>
<th>sales_dollar_amount</th>
<th>transaction_type</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>252</td>
<td>purchase</td>
<td>Inistar</td>
</tr>
<tr>
<td>363</td>
<td>purchase</td>
<td>Inistar</td>
</tr>
<tr>
<td>510</td>
<td>purchase</td>
<td>Inistar</td>
</tr>
<tr>
<td>-276</td>
<td>return</td>
<td>Foodcorp</td>
</tr>
<tr>
<td>252</td>
<td>purchase</td>
<td>Foodcorp</td>
</tr>
<tr>
<td>195</td>
<td>purchase</td>
<td>Foodcorp</td>
</tr>
<tr>
<td>290</td>
<td>purchase</td>
<td>Foodcorp</td>
</tr>
<tr>
<td>222</td>
<td>purchase</td>
<td>Foodcorp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foodgen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Goldcare</td>
</tr>
</tbody>
</table>

(10 rows)

**Controlling Join Inputs**

By default, the optimizer uses its own internal logic to determine whether to join one table to another as an inner or outer input. Occasionally, the optimizer might choose the larger table as the inner input to a join. Doing so can incur performance and concurrency issues.

If the configuration parameter `EnableForceOuter` is set to 1, you can control join inputs for specific tables through `ALTER TABLE..FORCE OUTER`. The
FORCE OUTER option modifies a table's force_outer setting in the system table TABLES. When implementing a join, Vertica compares the force_outer settings of the participating tables:

- If table settings are unequal, Vertica uses them to set the join inputs:
  - A table with a low force_outer setting relative to other tables is joined to them as an inner input.
  - A table with a high force_outer setting relative to other tables is joined to them as an outer input.
- If all table settings are equal, Vertica ignores them and assembles the join on its own.

The force_outer column is initially set to 5 for all newly-defined tables. You can use ALTER TABLE..FORCE OUTER to reset force_outer to a value equal to or greater than 0. For example, you might change the force_outer settings of tables abc and xyz to 3 and 8, respectively:

```
=> ALTER TABLE abc FORCE OUTER 3;
=> ALTER TABLE xyz FORCE OUTER 8;
```

Given these settings, the optimizer joins abc as the inner input to any table with a force_outer value greater than 3. The optimizer joins xyz as the outer input to any table with a force_outer value less than 8.

**Projection Inheritance**

When you query a projection directly, it inherits the force_outer setting of its anchor table. The query then uses this setting when joined to another projection.

**Enabling Forced Join Inputs**

The configuration parameter EnableForceOuter determines whether Vertica uses a table's force_outer value to implement a join. By default, this parameter is set to 0, and forced join inputs are disabled. You can enable forced join inputs at session and database scopes, through ALTER SESSION and ALTER DATABASE, respectively:

```
=> ALTER SESSION SET EnableForceOuter = { 0 | 1 };
=> ALTER DATABASE db-name SET EnableForceOuter = { 0 | 1 };
```

If EnableForceOuter is set to 0, ALTER TABLE..FORCE OUTER statements return with this warning:
WARNING 0: Set configuration parameter EnableForceOuter for the current session or the database in order to use force_outer value

Viewing Forced Join Inputs

EXPLAIN-generated query plans indicate whether the configuration parameter EnableForceOuter is on. A join query might include tables whose force_outer settings are less or greater than the default value of 5. In this case, the query plan includes a Force outer level field for the relevant join inputs.

For example, the following query joins tables store.store_sales and public.products, where both tables have the same force_outer setting (5). EnableForceOuter is on, as indicated in the generated query plan:

```
=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
    FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
    WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;
```

EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
  |   Order: sales.store_key ASC, sales.sale_date ASC
  |   Execute on: All Nodes
  |   +----> JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (BROADCAST)(LOCAL ROUND ROBIN)
  |   |     Join Cond: (sales.product_key = products.product_key)
  |   |     Execute on: All Nodes
  |   |   +-- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
  |   |   |     Projection: store.store_sales_b0
  |   |   |     Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
  |   |   |     Filter: (sales.sale_date = '2014-12-01'::date)
  |   |   |     Execute on: All Nodes
  |   |   |   +-- Inner -> STORAGE ACCESS for products [Cost: 177, Rows: 60K (NO STATISTICS)] (PATH ID: 4)
  |   |   |   |     Projection: public.products_b0
  |   |   |   |     Materialize: products.product_key, products.product_description
  |   |   |   |   Execute on: All Nodes

The following ALTER TABLE statement resets the force_outer setting of public.products to 1:

```
=> ALTER TABLE public.products FORCE OUTER 1;
```

The regenerated query plan for the same join now includes a Force outer level field and specifies public.products as the inner input:

```
=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
    FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
    WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;
```
EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +----- JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (BROADCAST)(LOCAL ROUND ROBIN)
| | Join Cond: (sales.product_key = products.product_key)
| | Execute on: All Nodes
| | +- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: store.store_sales_b0
| | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | Filter: (sales.sale_date = '2014-12-01':date)
| | | Execute on: All Nodes
| | +- Inner -> STORAGE ACCESS for products [Cost: 177, Rows: 60K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: public.products_b0
| | | Force outer level: 1
| | | Materialize: products.product_key, products.product_description
| | | Execute on: All Nodes

If you change the force_outer setting of public.products to 8, Vertica creates a different query plan that specifies public.products as the outer input:

```sql
=> ALTER TABLE public.products FORCE OUTER 8;
ALTER TABLE

=> EXPLAIN SELECT s.store_key, p.product_description, s.sales_quantity, s.sale_date
FROM store.store_sales s JOIN public.products p ON s.product_key=p.product_key
WHERE s.sale_date='2014-12-01' ORDER BY s.store_key, s.sale_date;

EnableForceOuter is on
Access Path:
+-SORT [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +----- JOIN HASH [Cost: 5K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Inner (BROADCAST)
| | Join Cond: (sales.product_key = products.product_key)
| | Materialize at Output: products.product_description
| | Execute on: All Nodes
| | +- Outer -> STORAGE ACCESS for products [Cost: 20, Rows: 60K (NO STATISTICS)] (PATH ID: 3)
| | | Projection: public.products_b0
| | | Force outer level: 8
| | | Materialize: products.product_key
| | | Execute on: All Nodes
| | | Runtime Filter: (SIP1(HashJoin): products.product_key)
| | +- Inner -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
| | | Projection: store.store_sales_b0
| | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | Filter: (sales.sale_date = '2014-12-01':date)
| | | Execute on: All Nodes
```

Restrictions

Vertica ignores force_outer settings when it performs the following operations:
• Outer joins: Vertica generally respects OUTER JOIN clauses regardless of the force_outer settings of the joined tables.

• MERGE statement joins.

• Queries that include the SYNTACTIC_JOIN hint.

• Half-join queries such as SEMI JOIN.

• Joins to subqueries, where the subquery is always processed as having a force_outer setting of 5 regardless of the force_outer settings of the tables that are joined in the subquery. This setting determines a subquery's designation as inner or outer input relative to other join inputs. If two subqueries are joined, the optimizer determines which one is the inner input, and which one the outer.

**Range Joins**

Vertica provides performance optimizations for <, <=, >, >=, and BETWEEN predicates in join ON clauses. These optimizations are particularly useful when a column from one table is restricted to be in a range specified by two columns of another table.

**Key Ranges**

Multiple, consecutive key values can map to the same dimension values. Consider, for example, a table of IPv4 addresses and their owners. Because large subnets (ranges) of IP addresses could belong to the same owner, this dimension can be represented as:

```sql
=> CREATE TABLE ipOwners(
    ip_start INTEGER,
    ip_end INTEGER,
    owner_id INTEGER);
=> CREATE TABLE clicks(
    ipowner INTEGER,
    dest_ip INTEGER);
```

A query that associates a click stream with its destination can use a join similar to the following, which takes advantage of the range optimization:

```sql
=> SELECT owner_id, COUNT(*) FROM clicks JOIN ipOwners
    ON clicks.dest_ip BETWEEN ip_start AND ip_end
    GROUP BY owner_id;
```
Requirements

Operators <, <=, >, >=, or BETWEEN must appear as top-level conjunctive predicates for range join optimization to be effective, as shown in the following examples:

**BETWEEN** as the only predicate:

=> SELECT COUNT(*) FROM fact JOIN dim
   ON fact.point BETWEEN dim.start AND dim.end;

**Comparison operators as top-level predicates (within AND):**

=> SELECT COUNT(*) FROM fact JOIN dim
   ON fact.point > dim.start AND fact.point < dim.end;

**BETWEEN** as a top-level predicate (within AND):

=> SELECT COUNT(*) FROM fact JOIN dim
   ON (fact.point BETWEEN dim.start AND dim.end) AND fact.c <> dim.c;

**Query not optimized because OR is top-level predicate (disjunctive):**

=> SELECT COUNT(*) FROM fact JOIN dim
   ON (fact.point BETWEEN dim.start AND dim.end) OR dim.end IS NULL;

Notes

- Expressions are optimized in range join queries in many cases.

- If range columns can have NULL values indicating that they are open-ended, it is possible to use range join optimizations by replacing nulls with very large or very small values:

=> SELECT COUNT(*) FROM fact JOIN dim
   ON fact.point BETWEEN NVL(dim.start, -1) AND NVL(dim.end, 100000000000);

- If there is more than one set of ranging predicates in the same ON clause, the order in which the predicates are specified might impact the effectiveness of the optimization:

=> SELECT COUNT(*) FROM fact JOIN dim
   ON fact.point1 BETWEEN dim.start1 AND dim.end1
   AND fact.point2 BETWEEN dim.start2 AND dim.end2;
The optimizer chooses the first range to optimize, so write your queries so that the range you most want optimized appears first in the statement.

- The use of the range join optimization is not directly affected by any characteristics of the physical schema; no schema tuning is required to benefit from the optimization.

- The range join optimization can be applied to joins without any other predicates, and to HASH or MERGE joins.

- To determine if an optimization is in use, search for RANGE in the EXPLAIN plan.
Query Optimization

When you submit a query to Vertica for processing, the Vertica query optimizer automatically chooses a set of operations to compute the requested result. These operations together are called a query plan. The choice of operations can significantly affect how many resources are needed to compute query results, and overall run-time performance. Optimal performance depends in great part on the projections that are available for a given query.

This section describes the different operations that the optimizer uses and how you can facilitate optimizer performance.

Note: Database response time depends on many factors. These include type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if response times are already short, or are not hardware bound.
Initial Process for Improving Query Performance

To optimize query performance, begin by performing the following tasks:

1. Run Database Designer.
2. Check query events proactively.
3. Review the query plan.

Run Database Designer

Database Designer creates a physical schema for your database that provides optimal query performance. The first time you run Database Designer, you should create a comprehensive design that includes relevant sample queries and data on which to base the design. If you develop performance issues later, consider loading additional queries that you run frequently and then rerunning Database Designer to create an incremental design.

When you run Database Designer, choose the option, Update Statistics. The Vertica query optimizer uses statistics about the data to create a query plan. Statistics help the optimizer determine:

- Multiple eligible projections to answer the query
- The best order in which to perform joins
- Data distribution algorithms, such as broadcast and resegmentation

If your statistics become out of date, run the Vertica function `ANALYZE_STATISTICS` function to update statistics for a schema, table, or columns. For more information, see Collecting Database Statistics.

Check Query Events Proactively

The `QUERY_EVENTS` system table returns information on query planning, optimization, and execution events.

The `EVENT_TYPE` column provides various event types:
Some event types are **informational**.

Others you should **review for possible corrective action**.

Several are **most important to address**.

### Review the Query Plan

A *query plan* is a sequence of step-like paths that the Vertica query optimizer selects to access or alter information in your Vertica database. There are two ways to get information about the query plan:

- Run the **EXPLAIN** command. Each step (path) represents a single operation that the optimizer uses for its execution strategy.

- Query the **QUERY_PLAN_PROFILES** system table. This table provides detailed execution status for currently running queries. Output from the **QUERY_PLAN_PROFILES** table shows the real-time flow of data and the time and resources consumed for each path in each query plan.

### See Also

- **QUERY_EVENTS**
Column Encoding

You can potentially make queries faster by changing column encoding. Encoding reduces the on-disk size of your data so the amount of I/O required for queries is reduced, resulting in faster execution times. Make sure all columns and projections included in the query use the correct data encoding. To do this, take the following steps:

1. Run Database Designer to create an incremental design. Database Designer implements the optimum encoding and projection design.

2. After creating the incremental design, update statistics using the ANALYZE_STATISTICS function.

3. Run EXPLAIN with one or more of the queries you submitted to the design to make sure it is using the new projections.

Alternatively, run DESIGNER_DESIGN_PROJECTION_ENCODINGS to re-evaluate the current encoding and update it if necessary.

Improving Column Compression

If you see slow performance or a large storage footprint with your FLOAT data, evaluate the data and your business needs to see if it can be contained in a NUMERIC column with a precision of 18 digits or less. Converting a FLOAT column to a NUMERIC column can improve data compression, reduce the on-disk size of your database, and improve performance of queries on that column.

When you define a NUMERIC data type, you specify the precision and the scale; NUMERIC data are exact representations of data. FLOAT data types represent variable precision and approximate values; they take up more space in the database.

Converting FLOAT columns to NUMERIC columns is most effective when:

- NUMERIC precision is 18 digits or less. Performance of NUMERIC data is fine-tuned for the common case of 18 digits of precision. Vertica recommends converting FLOAT columns to NUMERIC columns only if they require precision of 18 digits or less.

- FLOAT precision is bounded, and the values will all fall within a specified precision for a NUMERIC column. One example is monetary values like product prices or financial transaction amounts. For example, a column defined as NUMERIC(11,2) can accommodate
prices from 0 to a few million dollars and can store cents, and compresses more efficiently than a FLOAT column.

If you try to load a value into a NUMERIC column that exceeds the specified precision, Vertica returns an error and does not load the data. If you assign a value with more decimal digits than the specified scale, the value is rounded to match the specified scale and stored in that column.

See Also

Numeric Data Types

Using Run Length Encoding

When you run Database Designer, you can choose to optimize for loads, which minimizes database footprint. In this case, Database Designer applies encodings to columns to maximize query performance. Encoding options include run length encoding (RLE), which replaces sequences (runs) of identical values in a column with a set of pairs, where each pair represents the number of contiguous occurrences for a given value: \((\text{occurrences}, \text{value})\).

RLE is generally applicable to a column with low-cardinality, and where identical values are contiguous—typically, because table data is sorted on that column. For example, a customer profile table typically includes a gender column that contains values of F and M only. Sorting on gender ensures runs of F or M values that can be expressed as a set of two pairs: \((\text{occurrences}, \text{F})\) and \((\text{occurrences}, \text{M})\). So, given 8,147 occurrences of F and 7,956 occurrences of M, and a projection that is sorted primarily on gender, Vertica can apply RLE and store these values as a single set of two pairs: \((8147, \text{F})\) and \((7956, \text{M})\). Doing so reduces this projection’s footprint and improves query performance.

Projections for Queries with Predicates

If your query contains one or more predicates, you can modify the projections to improve the query’s performance, as described in the following two examples.
Queries That Use Date Ranges

This example shows how to encode data using RLE and change the projection sort order to improve the performance of a query that retrieves all data within a given date range.

Suppose you have a query that looks like this:

```sql
=> SELECT * FROM trades
    WHERE trade_date BETWEEN '2016-11-01' AND '2016-12-01';
```

To optimize this query, determine whether all of the projections can perform the SELECT operation in a timely manner. Run SELECT COUNT(*) statement for each projection, specifying the date range, and note the response time. For example:

```sql
=> SELECT COUNT(*) FROM [ projection_name ]
    WHERE trade_date BETWEEN '2016-11-01' AND '2016-12-01';
```

If one or more of the queries is slow, check the uniqueness of the trade_date column and determine if it needs to be in the projection’s ORDER BY clause and/or can be encoded using RLE. RLE replaces sequences of the same data values within a column by a pair that represents the value and a count. For best results, order the columns in the projection from lowest cardinality to highest cardinality, and use RLE to encode the data in low-cardinality columns.

**Note:** For an example of using sorting and RLE, see [Combine RLE and Sort Order](#).

If the number of unique columns is unsorted, or if the average number of repeated rows is less than 10, trade_date is too close to being unique and cannot be encoded using RLE. In this case, add a new column to minimize the search scope.

The following example adds a new column trade_year:

1. **Determine if the new column trade_year returns a manageable result set.** The following query returns the data grouped by trade_year:

```sql
=> SELECT DATE_TRUNC('trade_year', trade_date), COUNT(*)
    FROM trades
    GROUP BY DATE_TRUNC('trade_year', trade_date);
```

2. **Assuming that trade_year = 2007 is near 8k,** add a column for trade_year to the trades table. The SELECT statement then becomes:

```sql
=> SELECT * FROM trades
    WHERE trade_year = 2007
```
AND trade_date BETWEEN '2016-11-01' AND '2016-12-01';

As a result, you have a projection that is sorted on trade_year, which can be encoded using RLE.

Queries for Tables with a High-Cardinality Primary Key

This example demonstrates how you can modify the projection to improve the performance of queries that select data from a table with a high-cardinality primary key.

Suppose you have the following query:

```sql
=> SELECT FROM [table]
    WHERE pk IN (12345, 12346, 12347,...);
```

Because the primary key is a high-cardinality column, Vertica has to search a large amount of data.

To optimize the schema for this query, create a new column named buckets and assign it the value of the primary key divided by 10000. In this example, buckets=(int) pk/10000. Use the buckets column to limit the search scope as follows:

```sql
=> SELECT FROM [table]
    WHERE buckets IN (1,...)
    AND pk IN (12345, 12346, 12347,...);
```

Creating a lower cardinality column and adding it to the query limits the search scope and improves the query performance. In addition, if you create a projection where buckets is first in the sort order, the query may run even faster.
GROUP BY Queries

The following sections include several examples that show how you can design your projections to optimize the performance of your GROUP BY queries.

GROUP BY Implementation Options

Vertica implements a query GROUP BY clause with one of these algorithms: GROUPBY PIPELINED or GROUPBY HASH. Both algorithms return the same results. Performance of both is generally similar for queries that return a small number of distinct groups—typically a thousand per node.

You can use EXPLAIN to determine which algorithm the query optimizer chooses for a given query. The following conditions generally determine which algorithm is chosen:

- GROUPBY PIPELINED requires all GROUP BY data to be specified in the projection's ORDER BY clause. For details, see GROUPBY PIPELINED Requirements below.

  Because GROUPBY PIPELINED only needs to retain in memory the current group data, this algorithm generally requires less memory and executes faster than GROUPBY HASH. Performance improvements are especially notable for queries that aggregate large numbers of distinct groups.

- GROUPBY HASH is used for any query that does not comply with GROUP BY PIPELINED sort order requirements. In this case, Vertica must build a hash table on GROUP BY columns before it can start grouping the data.

GROUPBY PIPELINED Requirements

You can enable use of the GROUP BY PIPELINED algorithm by ensuring that the query and one of its projections comply with GROUP BY PIPELINED requirements. The following conditions apply to GROUPBY PIPELINED. If none of them is true for the query, then Vertica uses GROUPBY HASH.

All examples that follow assume this schema:

```sql
CREATE TABLE sortopt (  
a INT NOT NULL,  
b INT NOT NULL,)
```
c INT,
d INT
);
CREATE PROJECTION sortopt_p (  
a_proj,
b_proj,
c_proj,
d_proj )
AS SELECT * FROM sortopt
ORDER BY a,b,c
UNSEGMENTED ALL NODES;
INSERT INTO sortopt VALUES(5,2,13,84);
INSERT INTO sortopt VALUES(14,22,8,115);
INSERT INTO sortopt VALUES(79,9,401,33);

Condition 1

All GROUP BY columns are also included in the projection ORDER BY clause.

For example:

<table>
<thead>
<tr>
<th>GROUP BY columns</th>
<th>GROUPBY algorithm</th>
<th>Reason chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a,b</td>
<td>GROUPBY PIPELINED</td>
<td>Columns a, b, and c are included in the projection sort columns.</td>
</tr>
<tr>
<td>b,a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a,b,c</td>
<td>GROUPBY PIPELINED</td>
<td></td>
</tr>
<tr>
<td>c,a,b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a,b,c,d</td>
<td>GROUPBY HASH</td>
<td>Column d is not part of the projection sort columns.</td>
</tr>
</tbody>
</table>

Condition 2

If the query's GROUP BY clause has fewer columns than the projection's ORDER BY clause, the GROUP BY columns must:

- Be a subset of ORDER BY columns that are contiguous.
- Include the first ORDER BY column.

For example:

<table>
<thead>
<tr>
<th>GROUP BY columns</th>
<th>GROUPBY algorithm</th>
<th>Reason chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>GROUPBY PIPELINED</td>
<td>All GROUP BY columns are a subset of contiguous columns in the projection's ORDER BY clause {a, b, c}, and include column a.</td>
</tr>
<tr>
<td>a,b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b,a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP BY columns</td>
<td>GROUPBY algorithm</td>
<td>Reason chosen</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>a, c</td>
<td>GROUPBY HASH</td>
<td>GROUP BY columns {a, c} are not contiguous in the projection ORDER BY clause {a, b, c}.</td>
</tr>
<tr>
<td>b, c</td>
<td>GROUPBY HASH</td>
<td>GROUP BY columns {b, c} do not include the projection’s first ORDER BY column a.</td>
</tr>
</tbody>
</table>

**Condition 3**

If a query’s GROUP BY columns do not appear first in the projection’s ORDER BY clause, then any early-appearing projection sort columns that are missing in the query’s GROUP BY clause must be present as single-column constant equality predicates in the query’s WHERE clause.

For example:

<table>
<thead>
<tr>
<th>Query</th>
<th>GROUPBY algorithm</th>
<th>Reason chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY b</td>
<td>GROUPBY PIPELINED</td>
<td>All columns preceding b in the projection sort order appear as constant equality predicates.</td>
</tr>
<tr>
<td>SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY a, b</td>
<td>GROUPBY PIPELINED</td>
<td>Grouping column a is redundant but has no effect on algorithm selection.</td>
</tr>
<tr>
<td>SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY b, c</td>
<td>GROUPBY PIPELINED</td>
<td>All columns preceding b and c in the projection sort order appear as constant equality predicates.</td>
</tr>
<tr>
<td>SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY c, b</td>
<td>GROUPBY PIPELINED</td>
<td>All columns preceding b and c in the projection sort order appear as constant equality predicates.</td>
</tr>
<tr>
<td>SELECT SUM(a) FROM sortopt WHERE a = 10 GROUP BY c</td>
<td>GROUPBY HASH</td>
<td>All columns preceding c in the projection sort order do not appear as constant equality predicates.</td>
</tr>
</tbody>
</table>

**Controlling GROUPBY Algorithm Choice**

It is generally best to allow Vertica to determine which GROUP BY algorithm is best suited for a given query. Occasionally, you might want to use one algorithm over another. In such cases,
you can qualify the GROUP BY clause with a **GBTYTYPE** hint:

GROUP BY **/++ GBTYTYPE(HASH | PIPE ) */**

For example, given the following query, the query optimizer uses the GROUPBY PIPELINED algorithm:

```
=> EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY a,b;
          ---------------
QUERY PLAN DESCRIPTION:
          ---------------
EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY a,b;
Access Path:
  +GROUPBY PIPELINED [Cost: 11, Rows: 3 (NO STATISTICS)] (PATH ID: 1)
    | Aggregates: sum(sortopt.a)
    | Group By: sortopt.a, sortopt.b
...
```

You can use the GBTYTYPE hint to force the query optimizer to use the GROUPBY HASH algorithm instead:

```
=> EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY **/++GBTYTYPE(HASH) */** a,b;
          ---------------
QUERY Plan Description:
          ---------------
EXPLAIN SELECT SUM(a) FROM sortopt GROUP BY **/++GBTYTYPE(HASH) */** a,b;
Access Path:
  +GROUPBY HASH (LOCAL RESSEGMENT GROUPS) [Cost: 11, Rows: 3 (NO STATISTICS)] (PATH ID: 1)
    | Aggregates: sum(sortopt.a)
    | Group By: sortopt.a, sortopt.b
...
```

The GBTYTYPE hint can specify a PIPE (GROUPBY PIPELINED algorithm) argument only if the query and one of its projections comply with GROUP BY PIPELINED requirements. Otherwise, Vertica issues a warning and uses GROUPBY HASH.

For example, the following query cannot use the GROUPBY PIPELINED algorithm, as the GROUP BY columns \{b, c\} do not include the projection's first ORDER BY column a:

```
=> SELECT SUM(a) FROM sortopt GROUP BY **/++GBTYTYPE(PIPE) */** b,c;
          SUM
----
  79
 14
  5
(3 rows)
```
Avoiding Resegmentation During GROUP BY Optimization with Projection Design

To compute the correct result of a query that contains a GROUP BY clause, Vertica must ensure that all rows with the same value in the GROUP BY expressions end up at the same node for final computation. If the projection design already guarantees the data is segmented by the GROUP BY columns, no resegmentation is required at run time.

To avoid resegmentation, the GROUP BY clause must contain all the segmentation columns of the projection, but it can also contain other columns.

When your query includes a GROUP BY clause and joins, if the join depends on the results of the GROUP BY, as in the following example, Vertica performs the GROUP BY first:

```sql
=> EXPLAIN SELECT * FROM (SELECT b from foo GROUP BY b) AS F, foo WHERE foo.a = F.b;
```

Access Path:
+-JOIN MERGEJOIN(inputs presorted) [Cost: 649, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
  | Join Cond: (foo.a = F.b)
  | Materialize at Output: foo.b
  | Execute on: All Nodes
  +-- Outer -> STORAGE ACCESS for foo [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
  |   | Projection: public.foo_super
  |   | Materialize: foo.a
  |   | Execute on: All Nodes
  |   | Runtime Filter: (SIP1(MergeJoin): foo.a)
  |   +-- Inner -> SELECT [Cost: 245, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
  |   | Execute on: All Nodes
  |   | +--> GROUPBY HASH (SORT OUTPUT) (GLOBAL RESEGMENT GROUPS) (LOCAL RESEGMENT GROUPS) [Cost: 245, Rows: 10K (NO STATISTICS)] (PATH ID: 4)
  |   |   | Group By: foo.b
  |   |   | Execute on: All Nodes
  |   |   +-- STORAGE ACCESS for foo [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 5)
  |   |   |   | Projection: public.foo_super
  |   |   |   | Materialize: foo.b
  |   |   |   | Execute on: All Nodes

If the result of the join operation is the input to the GROUP BY clause, Vertica performs the join first, as in the following example. The segmentation of those intermediate results may not be consistent with the GROUP BY clause in your query, resulted in resegmentation at run time.

```sql
=> EXPLAIN SELECT * FROM foo AS F, foo WHERE foo.a = F.b GROUP BY 1,2,3,4;
```

Access Path:
+GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 869, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
 | Group By: F.a, F.b, foo.a, foo.b
 | Execute on: All Nodes
 | +-- JOIN HASH [Cost: 853, Rows: 10K (NO STATISTICS)] (PATH ID: 2) Outer (RESEGMENT)(LOCAL ROUND ROBIN)
 |   | Join Cond: (foo.a = F.b)
If your query does not include joins, the GROUP BY clauses are processed using the existing database projections.

**Examples**

Assume the following projection:

```
CREATE PROJECTION _ SEGMENTED BY HASH(a,b) ALL NODES
```

The following table explains whether or not resegmentation occurs at run time and why.

<table>
<thead>
<tr>
<th>GROUP BY a</th>
<th>Requires resegmentation at run time. The query does not contain all the projection segmentation columns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP BY a, b</td>
<td>Does not require resegmentation at run time. The GROUP BY clause contains all the projection segmentation columns.</td>
</tr>
<tr>
<td>GROUP BY a, b, c</td>
<td>Does not require resegmentation at run time. The GROUP BY clause contains all the projection segmentation columns.</td>
</tr>
<tr>
<td>GROUP BY a+1, b</td>
<td>Requires resegmentation at run time because of the expression on column a.</td>
</tr>
</tbody>
</table>

To determine if resegmentation will occur during your GROUP BY query, look at the EXPLAIN-generated query plan.

For example, the following plan uses GROUPBY PIPELINED sort optimization and requires resegmentation to perform the GROUP BY calculation:

```
+-GROUPBY PIPELINED (RESEGMENT GROUPS) [Cost: 194, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
```

The following plan uses GROUPBY PIPELINED sort optimization, but does not require resegmentation:

```
+-GROUPBY PIPELINED [Cost: 459, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
```
**DISTINCT in a SELECT Query List**

This section describes how to optimize queries that have the DISTINCT keyword in their SELECT list. The techniques for optimizing DISTINCT queries are similar to the techniques for optimizing GROUP BY queries because when processing queries that use DISTINCT, the Vertica optimizer rewrites the query as a GROUP BY query.

The following sections below this page describe specific situations:

- Query Has No Aggregates in SELECT List
- `COUNT (DISTINCT)` and Other DISTINCT Aggregates
- Approximate Count Distinct Functions
- Single DISTINCT Aggregates
- Multiple DISTINCT Aggregates

Examples in these sections use the following table:

```
=> CREATE TABLE table1 (  
a INT,  
b INT,  
c INT  
);
```

**Query Has No Aggregates in SELECT List**

If your query has no aggregates in the SELECT list, internally, Vertica treats the query as if it uses GROUP BY instead.

For example, you can rewrite the following query:

```
SELECT DISTINCT a, b, c FROM table1;
```

as:

```
SELECT a, b, c FROM table1 GROUP BY a, b, c;
```

For fastest execution, apply the optimization techniques for GROUP BY queries described in GROUP BY Queries.
COUNT (DISTINCT) and Other DISTINCT Aggregates

Computing a DISTINCT aggregate generally requires more work than other aggregates. Also, a query that uses a single DISTINCT aggregate consumes fewer resources than a query with multiple DISTINCT aggregates.

Tip: Vertica executes queries with multiple distinct aggregates more efficiently when all distinct aggregate columns have a similar number of distinct values.

Approximate Count Distinct Functions

Vertica provides the COUNT(DISTINCT) function to compute the exact number of distinct values in a data set. If projections are available that allow COUNT(DISTINCT) to execute using the GROUPBY PIPELINED algorithm, COUNT(DISTINCT) performs well. In some situations, however, using APPROXIMATE_COUNT_DISTINCT performs better than COUNT(DISTINCT).

A COUNT [Aggregate] operation performs well when:

- One of the sorted projections delivers an order that enables sorted aggregation to be performed.
- The number of distinct values is fairly small.
- Hashed aggregation is required to execute the query.

When an approximate value will suffice or the values need to be rolled up, consider using the APPROXIMATE_COUNT_DISTINCT* functions.

Note: The APPROXIMATE_COUNT_DISTINCT* functions cannot appear in the same query block as DISTINCT aggregates.

Use Cases

Use APPROXIMATE_COUNT_DISTINCT as a direct replacement for COUNT (DISTINCT) when:

- You have a large data set and you do not require an exact count of distinct values.
- The performance of COUNT(DISTINCT) on a given data set is insufficient.
- You calculate several distinct counts in the same query.
- The plan for COUNT(DISTINCT) uses hashed aggregation.

Most of the time, APPROXIMATE_COUNT_DISTINCT executes faster than a comparable COUNT (DISTINCT) operation when hashed.

The expected value that APPROXIMATE_COUNT_DISTINCT returns is equal to COUNT (DISTINCT), with an error that is lognormally distributed with standard deviation s. You can control the standard deviation directly by setting the error_tolerance.

Use APPROXIMATE_COUNT_DISTINCT_SYNOPSIS and APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS together when:

- You have a large data set and you don't require an exact count of distinct values.
- The performance of COUNT(DISTINCT) on a given data set is insufficient.
  
  and

- You want to pre-compute the distinct counts and later combine them in different ways.

Pass APPROXIMATE_COUNT_DISTINCT_SYNOPSIS the data set and a normally distributed confidence interval. The function returns a subset of the data, called a synopsis.

Pass the synopsis to the APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS function, which then performs an approximate count distinct on the synopsis.

### Single DISTINCT Aggregates

Vertica computes a DISTINCT aggregate by first removing all duplicate values of the aggregate's argument to find the distinct values. Then it computes the aggregate.

For example, you can rewrite the following query:

```
SELECT a, b, COUNT(DISTINCT c) AS dcnt FROM table1 GROUP BY a, b;
```

as:

```
SELECT a, b, COUNT(dcnt) FROM
  (SELECT a, b, c AS dcnt FROM table1 GROUP BY a, b, c)
GROUP BY a, b;
```

For fastest execution, apply the optimization techniques for GROUP BY queries.
Multiple DISTINCT Aggregates

If your query has multiple DISTINCT aggregates, there is no straightforward SQL rewrite that can compute them. The following query cannot easily be rewritten for improved performance:

```
SELECT a, COUNT(DISTINCT b), COUNT(DISTINCT c) AS dcnt FROM table1 GROUP BY a;
```

For a query with multiple DISTINCT aggregates, there is no projection design that can avoid using GROUPBY HASH and resegmenting the data. To improve performance of this query, make sure that it has large amounts of memory available. For more information about memory allocation for queries, see Resource Manager.
JOIN Queries

In general, you can optimize execution of queries that join multiple tables in several ways:

- Create projections for the joined tables that are sorted on join predicate columns. This facilitates use of the merge join algorithm, which generally joins tables more efficiently than the merge join algorithm.

- Create projections that are identically segmented on the join keys.

Other Best Practices

Vertica also executes joins more efficiently if the following conditions are true:

- Query construction enables the query optimizer to create a plan where the larger table is defined as the outer input.

- The columns on each side of the equality predicate are from the same table. For example in the following query, the left and right sides of the equality predicate include only columns from tables T and X, respectively:

  ```sql
  => SELECT * FROM T JOIN X ON T.a + T.b = X.x1 - X.x2;
  ```

  Conversely, the following query incurs more work to process, because the right side of the predicate includes columns from both tables T and X:

  ```sql
  => SELECT * FROM T JOIN X WHERE T.a = X.x1 + T.b
  ```

Hash Joins Versus Merge Joins

The Vertica optimizer implements a join with one of the following algorithms:

- Merge join is used when projections of the joined tables are sorted on the join columns. Merge joins are faster and uses less memory than hash joins.

- Hash join is used when projections of the joined tables are not already sorted on the join columns. In this case, the optimizer builds an in-memory hash table on the inner table's join column. The optimizer then scans the outer table for matches to the hash table, and joins
data from the two tables accordingly. The cost of performing a hash join is low if the entire hash table can fit in memory. Cost rises significantly if the hash table must be written to disk.

The optimizer automatically chooses the most appropriate algorithm to execute a query, given the projections that are available.

**Facilitating Merge Joins**

To facilitate a merge join, create projections for the joined tables that are sorted on the join predicate columns. The join predicate columns should be the first columns in the ORDER BY clause.

For example, tables `first` and `second` are defined as follows, with projections `first_p1` and `second_p1`, respectively. The projections are sorted on `data_first` and `data_second`:

```sql
CREATE TABLE first (id INT, data_first INT);
CREATE PROJECTION first_p1 AS SELECT * FROM first ORDER BY data_first;

CREATE TABLE second (id INT, data_second INT);
CREATE PROJECTION second_p1 AS SELECT * FROM first ORDER BY data_second;
```

When you join these tables on unsorted columns `first.id` and `second.id`, Vertica uses the hash join algorithm:

```
EXPLAIN SELECT first.data_first, second.data_second FROM first JOIN second ON first.id = second.id;
Access Path:
+JOIN HASH [Cost: 752, Rows: 300K] (PATH ID: 1) Inner (BROADCAST)
```

You can facilitate execution of this query with the merge join algorithm by creating projections `first_p2` and `second_p2`, which are sorted on join columns `first_p2.id` and `second_p2.id`, respectively:

```sql
CREATE PROJECTION first_p2 AS id, data_first FROM first ORDER BY id SEGMENTED BY hash(id, data_first) ALL NODES;
CREATE PROJECTION second_p2 AS id, data_second FROM second ORDER BY id SEGMENTED BY hash(id, data_second) ALL NODES;
```

If the query joins significant amounts of data, the query optimizer uses the merge algorithm:

```
EXPLAIN SELECT first.data_first, second.data_second FROM first JOIN second ON first.id = second.id;
Access Path:
+JOIN MERGEJOIN(inputs presorted) [Cost: 731, Rows: 300K] (PATH ID: 1) Inner (BROADCAST)
```

You can also facilitate a merge join by using subqueries to pre-sort the join predicate columns. For example:
Identical Segmentation

To improve query performance when you join multiple tables, create projections that are identically segmented on the join keys. Identically-segmented projections allow the joins to occur locally on each node, thereby helping to reduce data movement across the network during query processing.

To determine if projections are identically-segmented on the query join keys, create a query plan with EXPLAIN. If the query plan contains RESEGMENT or BROADCAST, the projections are not identically segmented.

The Vertica optimizer chooses a projection to supply rows for each table in a query. If the projections to be joined are segmented, the optimizer evaluates their segmentation against the query join expressions. It thereby determines whether the rows are placed on each node so it can join them without fetching data from another node.

Join Conditions for Identically Segmented Projections

A projection p is segmented on join columns if all column references in p’s segmentation expression are a subset of the columns in the join expression.

The following conditions must be true for two segmented projections p1 of table t1 and p2 of table t2 to participate in a join of t1 to t2:

- The join condition must have the following form:

  \[ t1.j1 = t2.j1 \text{ AND } t1.j2 = t2.j2 \text{ AND } ... \text{ AND } t1.jN = t2.jN \]

- The join columns must share the same base data type. For example:
  - If \( t1.j1 \) is an INTEGER, \( t2.j1 \) can be an INTEGER but it cannot be a FLOAT.
  - If \( t1.j1 \) is a CHAR(10), \( t2.j1 \) can be any CHAR or VARCHAR (for example, CHAR(10), VARCHAR(10), VARCHAR(20)), but \( t2.j1 \) cannot be an INTEGER.
  - If \( p1 \) is segmented by an expression on columns \( \{t1.s1, \ t1.s2, \ ... \ \ t1.sN\} \), each segmentation column \( t1.sX \) must be in the join column set \( \{t1.jX\} \).
If \( p_2 \) is segmented by an expression on columns \{ t_2.s_1, t_2.s_2, \ldots t_2.s_N \}, each segmentation column \( t_2.s_X \) must be in the join column set \{ t_2.j_X \}.

The segmentation expressions of \( p_1 \) and \( p_2 \) must be structurally equivalent. For example:

- If \( p_1 \) is SEGMENTED BY hash\((t_1.x)\) and \( p_2 \) is SEGMENTED BY hash\((t_2.x)\), \( p_1 \) and \( p_2 \) are identically segmented.
- If \( p_1 \) is SEGMENTED BY hash\((t_1.x)\) and \( p_2 \) is SEGMENTED BY hash\((t_2.x + 1)\), \( p_1 \) and \( p_2 \) are not identically segmented.

\( p_1 \) and \( p_2 \) must have the same segment count.

The assignment of segments to nodes must match. For example, if \( p_1 \) and \( p_2 \) use an OFFSET clause, their offsets must match.

If Vertica finds projections for \( t_1 \) and \( t_2 \) that are not identically segmented, the data is redistributed across the network during query run time, as necessary.

Tip: If you create custom designs, try to use segmented projections for joins whenever possible.

Examples

The following statements create two tables and specify to create identical segments:

```sql
=> CREATE TABLE t1 (id INT, x1 INT, y1 INT) SEGMENTED BY HASH(id, x1) ALL NODES;
=> CREATE TABLE t2 (id INT, x1 INT, y1 INT) SEGMENTED BY HASH(id, x1) ALL NODES;
```

Given this design, the join conditions in the following queries can leverage identical segmentation:

```sql
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.id;
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.id AND t1.x1 = t2.x1;
```

Conversely, the join conditions in the following queries require resegmentation:

```sql
=> SELECT * FROM t1 JOIN t2 ON t1.x1 = t2.x1;
=> SELECT * FROM t1 JOIN t2 ON t1.id = t2.x1;
```
See Also

- Partitioning and Segmentation
- CREATE PROJECTION
## ORDER BY Queries

You can improve the performance of queries that contain only ORDER BY clauses if the columns in a projection's ORDER BY clause are the same as the columns in the query.

If you define the projection sort order in the CREATE PROJECTION statement, the Vertica query optimizer does not have to sort projection data before performing certain ORDER BY queries.

The following table, sortopt, contains the columns a, b, c, and d. Projection sortopt_p specifies to order on columns a, b, and c.

```sql
CREATE TABLE sortopt (
   a INT NOT NULL,
   b INT NOT NULL,
   c INT,
   d INT
);
CREATE PROJECTION sortopt_p (
   a_proj,
   b_proj,
   c_proj,
   d_proj
) AS SELECT * FROM sortopt
ORDER BY a,b,c
UNSEGMENTED ALL NODES;
INSERT INTO sortopt VALUES(5,2,13,84);
INSERT INTO sortopt VALUES(14,22,8,115);
INSERT INTO sortopt VALUES(79,9,401,33);
```

Based on this sort order, if a SELECT * FROM sortopt query contains one of the following ORDER BY clauses, the query does not have to resort the projection:

- ORDER BY a
- ORDER BY a, b
- ORDER BY a, b, c

For example, Vertica does not have to resort the projection in the following query because the sort order includes columns specified in the CREATE PROJECTION .ORDER BY a, b, c clause, which mirrors the query's ORDER BY a, b, c clause:

```sql
=> SELECT * FROM sortopt ORDER BY a, b, c;
   |   |   |   |
---|---|---|---|
 5 | 2 | 13 | 84
14|22|8|115
```
If you include column d in the query, Vertica must re-sort the projection data because column d was not defined in the CREATE PROJECTION..ORDER BY clause. Therefore, the ORDER BY d query won't benefit from any sort optimization.

You cannot specify an ASC or DESC clause in the CREATE PROJECTION statement's ORDER BY clause. Vertica always uses an ascending sort order in physical storage, so if your query specifies descending order for any of its columns, the query still causes Vertica to re-sort the projection data. For example, the following query requires Vertica to sort the results:

```
=> SELECT * FROM sortopt ORDER BY a DESC, b, c;
```

```text
a | b | c | d
---+---+---+---
79 | 9 | 401 | 33
14 | 22 | 8 | 115
5 | 2 | 13 | 84
```

(3 rows)

See Also

CREATE PROJECTION
Analytic Functions

The following sections describe how to optimize SQL-99 analytic functions that Vertica supports.

Empty OVER Clauses

The OVER() clause does not require a windowing clause. If your query uses an analytic function like SUM(x) and you specify an empty OVER() clause, the analytic function is used as a reporting function, where the entire input is treated as a single partition; the aggregate returns the same aggregated value for each row of the result set. The query executes on a single node, potentially resulting in poor performance.

If you add a PARTITION BY clause to the OVER() clause, the query executes on multiple nodes, improving its performance.

NULL Sort Order

By default, projection column values are stored in ascending order, but placement of NULL values depends on a column's data type.

NULL Placement Differences With ORDER BY Clauses

The analytic OVER(window-order-clause) and the SQL ORDER BY clause have slightly different semantics:

OVER(ORDER BY ...)

The analytic window order clause uses the ASC or DESC sort order to determine NULLS FIRST or NULLS LAST placement for analytic function results. NULL values are placed as follows:

- ASC, NULLS LAST — NULL values appear at the end of the sorted result.
- DESC, NULLS FIRST — NULL values appear at the beginning of the sorted result.

(SQL) ORDER BY

The SQL and [[Undefined variable Vertica.DBMS_UPPERCASE]] ORDER BY clauses produce different results. The SQL ORDER BY clause specifies only ascending or descending sort order.
The ORDER BY clause determines NULL placement based on the column data type:

- NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL columns: NULLS FIRST (NULL values appear at the beginning of a sorted projection.)
- FLOAT, STRING, and BOOLEAN columns: NULLS LAST (NULL values appear at the end of a sorted projection.)

**NULL Sort Options**

If you do not care about NULL placement in queries that involve analytic computations, or if you know that columns do not contain any NULL values, specify NULLS AUTO—irrespective of data type. Vertica chooses the placement that gives the fastest performance, as in the following query. Otherwise, specify NULLS FIRST or NULLS LAST.

```sql
g=> SELECT x, RANK() OVER (ORDER BY x NULLS AUTO) FROM t;
```

You can carefully formulate queries so Vertica can avoid sorting the data and increase query performance, as illustrated by the following example. Vertica sorts inputs from table t on column x, as specified in the `OVER(ORDER BY)` clause, and then evaluates `RANK()`:

```sql
g=> CREATE TABLE t (  
x FLOAT,  
y FLOAT );  
g=> CREATE PROJECTION t_p (x, y) AS SELECT * FROM t  
    ORDER BY x, y UNSEGMENTED ALL NODES;  
g=> SELECT x, RANK() OVER (ORDER BY x) FROM t;
```

In the preceding SELECT statement, Vertica eliminates the ORDER BY clause and executes the query quickly because column x is a FLOAT data type. As a result, the projection sort order matches the analytic default ordering (ASC + NULLS LAST). Vertica can also avoid having to sort the data when the underlying projection is already sorted.

However, if column x is an INTEGER data type, Vertica must sort the data because the projection sort order for INTEGER data types (ASC + NULLS FIRST) does not match the default analytic ordering (ASC + NULLS LAST). To help Vertica eliminate the sort, specify the placement of NULLs to match the default ordering:

```sql
g=> SELECT x, RANK() OVER (ORDER BY x NULLS FIRST) FROM t;
```

If column x is a STRING, the following query eliminates the sort:

```sql
g=> SELECT x, RANK() OVER (ORDER BY x NULLS LAST) FROM t;
```
If you omit NULLS LAST in the preceding query, `[[[Undefined variable Vertica.DBMS_UPPERCASE]]]` eliminates the sort because `ASC + NULLS LAST` is the default sort specification for both the analytic `ORDER BY` clause and for string-related columns in Vertica.

See Also

- Runtime Sorting of NULL Values in Analytic Functions
- SQL Analytics

Runtime Sorting of NULL Values in Analytic Functions

By carefully writing queries or creating your design (or both), you can help the Vertica query optimizer skip sorting all columns in a table when performing an analytic function, which can improve query performance.

To minimize Vertica's need to sort projections during query execution, redefine the `employee` table and specify that NULL values are not allowed in the sort fields:

```
=> DROP TABLE employee CASCADE;
=> CREATE TABLE employee
  (empno INT,
   deptno INT NOT NULL,
   sal INT NOT NULL);
CREATE TABLE
=> CREATE PROJECTION employee_p AS
  SELECT * FROM employee
  ORDER BY deptno, sal;
CREATE PROJECTION
=> INSERT INTO employee VALUES(101,10,50000);
=> INSERT INTO employee VALUES(103,10,43000);
=> INSERT INTO employee VALUES(104,10,45000);
=> INSERT INTO employee VALUES(105,20,97000);
=> INSERT INTO employee VALUES(108,20,33000);
=> INSERT INTO employee VALUES(109,20,51000);
=> COMMIT;
COMMIT
```

```
=> SELECT * FROM employee;
  empno | deptno | sal
-------+--------+--------
   101  |    10  | 50000
   103  |    10  | 43000
   104  |    10  | 45000
   105  |    20  | 97000
   108  |    20  | 33000
   109  |    20  | 51000
(6 rows)
=> SELECT deptno, sal, empno, RANK() OVER
(PARTITION BY deptno ORDER BY sal)
FROM employee;

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>43000</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>45000</td>
<td>104</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>50000</td>
<td>101</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>33000</td>
<td>108</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>51000</td>
<td>109</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>97000</td>
<td>105</td>
<td>3</td>
</tr>
</tbody>
</table>

(6 rows)

Tip: If you do not care about NULL placement in queries that involve analytic computations, or if you know that columns contain no NULL values, specify NULLS AUTO in your queries. Vertica attempts to choose the placement that gives the fastest performance. Otherwise, specify NULLS FIRST or NULLS LAST.

LIMIT Queries with ROW_NUMBER Predicates

Queries that use the LIMIT clause with ORDER BY or analytic function ROW_NUMBER() return a specific subset of rows in the query result. Vertica processes these queries efficiently using Top-K Optimization, which is a database query ranking process. Top-K optimization avoids sorting (and potentially writing to disk) an entire data set to find a small number of rows. This can significantly improve query performance.

For example, in the following query, Vertica extracts only the three smallest rows from column x:

=> SELECT * FROM t1 ORDER BY x LIMIT 3;

If table t1 contains millions of rows, it is time consuming to sort all the x values. Instead, Vertica keeps track of the smallest three values in x.

Note: If you omit the ORDER BY clause, when using the LIMIT clause, the results can be nondeterministic.

Sort operations that precede a SQL analytics computation benefit from Top-K optimization if the query contains an OVER(ORDER BY) clause and a predicate on the ROW_NUMBER function, as in the following example:

=> SELECT x FROM
    (SELECT *, ROW_NUMBER() OVER (ORDER BY x) AS row
     FROM t1) t2 WHERE row <= 3;

The preceding query has the same behavior as the following query, which uses a LIMIT clause:
You can use \texttt{ROW\_NUMBER()} with the analytic \textit{window partition clause}, something you cannot do if you use \texttt{LIMIT}:

\begin{verbatim}
=> SELECT x, y FROM 
  (SELECT *, ROW\_NUMBER() OVER (PARTITION BY x ORDER BY y) AS row FROM t1) t2 WHERE row <= 3;
\end{verbatim}

\textbf{Note:} When the \texttt{OVER()} clause includes the \textit{window-particle-clause}, Top-K optimization occurs only when the analytic sort matches the input's sort, for example, if the projection is sorted on columns \texttt{x} and \texttt{y} in table \texttt{t1}.

If you still want to improve the performance of your query, consider using the optimization techniques described in \texttt{ORDER BY Queries}.
**INSERT-SELECT Operations**

There are several ways to optimize an INSERT-SELECT query that has the following format:

```sql
INSERT /*+direct*/ INTO destination SELECT * FROM source;
```

**Matching Sort Orders**

When performing INSERT-SELECT operations, to avoid the sort phase of the INSERT, make sure that the sort order for the SELECT query matches the projection sort order of the target table.

For example, on a single-node database:

```sql
=> CREATE TABLE source (col1 INT, col2 INT, col3 INT);
=> CREATE PROJECTION source_p (col1, col2, col3) 
   AS SELECT col1, col2, col3 FROM source 
   ORDER BY col1, col2, col3 
   SEGMENTED BY HASH(col3) 
   ALL NODES;
=> CREATE TABLE destination (col1 INT, col2 INT, col3 INT);
=> CREATE PROJECTION destination_p (col1, col2, col3) 
   AS SELECT col1, col2, col3 FROM destination 
   ORDER BY col1, col2, col3 
   SEGMENTED BY HASH(col3) 
   ALL NODES;
```

The following INSERT does not require a sort because the query result has the column order of the projection:

```sql
=> INSERT /*+direct*/ INTO destination SELECT * FROM source;
```

The following INSERT requires a sort because the order of the columns in the SELECT statement does not match the projection order:

```sql
=> INSERT /*+direct*/ INTO destination SELECT col1, col3, col2 FROM source;
```

The following INSERT does not require a sort. The order of the columns doesn't match, but the explicit ORDER BY causes the output to be sorted by `c1`, `c3`, `c2` in Vertica:

```sql
=> INSERT /*+direct*/ INTO destination SELECT col1, col3, col2 FROM source 
    GROUP BY col1, col3, col2 
    ORDER BY col1, col2, col3 ;
```
Identical Segmentation

When performing an INSERT-SELECT operation from a segmented source table to a segmented destination table, segment both projections on the same column to avoid resegmenting the data, as in the following example:

```sql
CREATE TABLE source (col1 INT, col2 INT, col3 INT);
CREATE PROJECTION source_p (col1, col2, col3) AS
  SELECT col1, col2, col3 FROM source
  SEGMENTED BY HASH(col3) ALL NODES;
CREATE TABLE destination (col1 INT, col2 INT, col3 INT);
CREATE PROJECTION destination_p (col1, col2, col3) AS
  SELECT col1, col2, col3 FROM destination
  SEGMENTED BY HASH(col3) ALL NODES;
INSERT /*+direct*/ INTO destination SELECT * FROM source;
```
DELETE and UPDATE Queries

Vertica is optimized for query-intensive workloads, so DELETE and UPDATE queries might not achieve the same level of performance as other queries. A DELETE and UPDATE operation has to update all projections, so the operation is as slow as the slowest projection. For additional information, see Using INSERT, UPDATE, and DELETE.

The topics that follow discuss best practices for optimizing DELETE and UPDATE queries in Vertica.

DELETE and UPDATE Performance Considerations

To improve the performance of your DELETE and UPDATE queries, consider the following issues:

- **Query performance after large deletes**—A large number of (unpurged) deleted rows can negatively affect query performance.

  To eliminate rows that have been deleted from the result, a query must do extra processing. If 10% or more of the total rows in a table have been deleted, the performance of a query on the table degrades. However, your experience may vary depending on the size of the table, the table definition, and the query. If a table has a large number of deleted rows, consider purging those rows to improve performance. For more information on purging, see Purging Deleted Data.

- **Recovery performance**—Recovery is the action required for a cluster to restore K-safety after a crash. Large numbers of deleted records can degrade the performance of a recovery. To improve recovery performance, purge the deleted rows. For more information on purging, see Purging Deleted Data.

- **Concurrency**—DELETE and UPDATE take exclusive locks on the table. Only one DELETE or UPDATE transaction on a table can be in progress at a time and only when no loads (or INSERTs) are in progress. DELETEs and UPDATEs on different tables can be run concurrently.

For detailed tips about improving DELETE and UPDATE performance, see DELETE and UPDATE Optimization.
Caution: Vertica does not remove deleted data immediately but keeps it as history for the purposes of historical query. A large amount of history can result in slower query performance. For information about how to configure the appropriate amount of history to retain, see Purging Deleted Data.

DELETE and UPDATE Optimization

The process of optimizing DELETE and UPDATE queries is the same for both operations. Some simple steps can increase the query performance by tens to hundreds of times. The following sections describe several ways to improve projection design and improve DELETE and UPDATE queries to significantly increase DELETE and UPDATE performance.

Note: For large bulk deletion, Vertica recommends using Partitioned Tables where possible because they provide the best DELETE performance and improve query performance.

Projection Column Requirements for Optimized Deletes

When all columns required by the DELETE or UPDATE predicate are present in a projection, the projection is optimized for DELETEs and UPDATEs. DELETE and UPDATE operations on such projections are significantly faster than on non-optimized projections.

For example, consider the following table and projections:

```sql
=> CREATE TABLE t (a INTEGER, b INTEGER, c INTEGER);
=> CREATE PROJECTION p1 (a, b, c) AS SELECT * FROM t ORDER BY a;
=> CREATE PROJECTION p2 (a, c) AS SELECT a, c FROM t ORDER BY c, a;
```

In the following query, both p1 and p2 are eligible for DELETE and UPDATE optimization because column ais available:

```sql
=> DELETE from t WHERE a = 1;
```

In the following example, only projection p1 is eligible for DELETE and UPDATE optimization because the b column is not available in p2:

```sql
=> DELETE from t WHERE b = 1;
```

Optimized Deletes in Subqueries

To be eligible for DELETE optimization, all target table columns referenced in a DELETE or UPDATE statement's WHERE clause must be in the projection definition.
For example, the following simple schema has two tables and three projections:

```sql
=> CREATE TABLE tb1 (a INT, b INT, c INT, d INT);
=> CREATE TABLE tb2 (g INT, h INT, i INT, j INT);
```

The first projection references all columns in tb1 and sorts on column a:

```sql
=> CREATE PROJECTION tb1_p AS SELECT a, b, c, d FROM tb1 ORDER BY a;
```

The buddy projection references and sorts on column a in tb1:

```sql
=> CREATE PROJECTION tb1_p_2 AS SELECT a FROM tb1 ORDER BY a;
```

This projection references all columns in tb2 and sorts on column i:

```sql
=> CREATE PROJECTION tb2_p AS SELECT g, h, i, j FROM tb2 ORDER BY i;
```

Consider the following DML statement, which references tb1.a in its WHERE clause. Since both projections on tb1 contain column a, both are eligible for the optimized DELETE:

```sql
=> DELETE FROM tb1 WHERE tb1.a IN (SELECT tb2.i FROM tb2);
```

Restrictions

Optimized DELETEs are not supported under the following conditions:

- With replicated projections if subqueries reference the target table. For example, the following syntax is not supported:

  ```sql
  => DELETE FROM tb1 WHERE tb1.a IN (SELECT e FROM tb2, tb2 WHERE tb2.e = tb1.e);
  ```

- With subqueries that do not return multiple rows. For example, the following syntax is not supported:

  ```sql
  => DELETE FROM tb1 WHERE tb1.a = (SELECT k FROM tb2);
  ```

Projection Sort Order for Optimizing Deletes

Design your projections so that frequently-used DELETE or UPDATE predicate columns appear in the sort order of all projections for large DELETEs and UPDATEs.

For example, suppose most of the DELETE queries you perform on a projection look like the following:
To optimize the DELETEs, make time_key appear in the ORDER BY clause of all your projections. This schema design results in better performance of the DELETE operation.

In addition, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. For more information, see Combine RLE and Sort Order. To analyze projections for sort order issues, use the EVALUATE_DELETE_PERFORMANCE function.

**Data Collector Table Queries**

The Vertica Data Collector extends system table functionality by gathering and retaining information about your database cluster. The Data Collector makes this information available in system tables.

Vertica Analytic Database stores Data Collection data in the Data Collector directory under the Vertica or catalog path. Use Data Collector information to query the past state of system tables and extract aggregate information.

In general, queries on Data Collector tables are more efficient when they include only the columns that contain the desired data. Queries are also more efficient when they:

- Avoid resegmentation
- Use time predicates

**Avoiding Resegmentation**

You can avoid resegmentation when you join the following DC tables on session_id or transaction_id, because all data is local:

- dc_session_starts
- dc_session_ends
- dc_requests_issued
- dc_requests_completed

Resegmentation is not required when a query includes the node_name column. For example:
This query runs efficiently because:

- The **initiator node** writes only to `dc_requestsIssued` and `dc_requestsCompleted`.
- Columns `session_id` and `nodeName` are correlated.

### Using Time Predicates

Use non-volatile functions and `TIMESTAMP` for the time range predicates. Vertica Analytic Database optimizes SQL performance for DC tables that use the time predicate.

Each DC table has a `time` column. Use this column to enter the time range as the query predicate.

For example, this query returns data for dates between September 1 and September 10:

```sql
SELECT *
FROM dc_foo
WHERE time > 'Sept 1, 2015::timestamp' AND time < 'Sept 10 2015'::timestamp;
```

You can change the minimum and maximum time values to adjust the time for which you want to retrieve data.

You must use non-volatile functions as time predicates. **Volatile functions** cause queries to run inefficiently. This example returns all queries that started and ended on April 7, 2015. However, the query runs at less than optimal performance because `trunc` and `timestamp` are volatile:

```sql
SELECT dri.transaction_id, dri.request, drc.processed_row_count
FROM dc_requestsIssued dri
LEFT JOIN dc_requestsCompleted drc
ON (dri.node_name, dri.session_id, dri.request_id)
WHERE trunc(dri.time, 'DD') > 'April 7,2015':timestamp
AND trunc(drc.time, 'DD') < 'April 8,2015':timestamp;
```
Views

A view is a stored query that encapsulates one or more SELECT statements. Views dynamically access and compute data from the database at execution time. A view is read-only, and can reference any combination of tables, temporary tables, and other views.

You can use views to achieve the following goals:

- Hide the complexity of SELECT statements from users for support or security purposes. For example, you can create a view that exposes only the data users need from various tables, while withholding sensitive data from the same tables.

- Encapsulate details about table structures, which might change over time, behind a consistent user interface.

Unlike projections, views are not materialized—that is, they do not store data on disk. Thus, the following restrictions apply:

- Vertica does not need to refresh view data when the underlying table data changes. However, a view does incur overhead to access and compute data.

- Views do not support inserts, deletes, or updates.

Creating Views

You can create two types of views:

- `CREATE VIEW` creates a view that persists across all sessions until it is explicitly dropped with `DROP VIEW`

- `CREATE LOCAL TEMPORARY VIEW` creates a view that is accessible only during the current Vertica session, and only to its creator. The view is automatically dropped when the current session ends.

After you create a view, you cannot change its definition. You can replace it with another view of the same name; or you can delete and redefine it.

Create Permissions

To create a view, you must be a superuser or have the following privileges:
<table>
<thead>
<tr>
<th>Privilege</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Schema where the view is created.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Tables and views referenced by the view query.</td>
</tr>
<tr>
<td>USAGE</td>
<td>All schemas that contain tables and views referenced by the view query.</td>
</tr>
</tbody>
</table>

For information about enabling users to access views, see Enabling View Access.

### Using Views

Views can be used in the FROM clause of any SQL query or subquery. At execution, Vertica internally substitutes the name of the view used in the query with the actual query used in the view definition.

### CREATE VIEW Example

The following `CREATE VIEW` statement creates the view `myview`, which sums all individual incomes of customers listed in the `store.store_sales_fact` table, and groups results by state:

```sql
=> CREATE VIEW myview AS
    SELECT SUM(annual_income), customer_state FROM public.customer_dimension
    WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact)
    GROUP BY customer_state
    ORDER BY customer_state ASC;
```

You can use this view to find all combined salaries greater than $2 billion:

```sql
=> SELECT * FROM myview where sum > 2000000000 ORDER BY sum DESC;

<table>
<thead>
<tr>
<th>SUM</th>
<th>customer_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>29253817091</td>
<td>CA</td>
</tr>
<tr>
<td>14215397659</td>
<td>TX</td>
</tr>
<tr>
<td>5225333668</td>
<td>MI</td>
</tr>
<tr>
<td>4907216137</td>
<td>CO</td>
</tr>
<tr>
<td>4581840709</td>
<td>IL</td>
</tr>
<tr>
<td>3769455689</td>
<td>CT</td>
</tr>
<tr>
<td>3330524215</td>
<td>FL</td>
</tr>
<tr>
<td>3310667307</td>
<td>IN</td>
</tr>
<tr>
<td>2832710696</td>
<td>TN</td>
</tr>
<tr>
<td>2806158503</td>
<td>PA</td>
</tr>
<tr>
<td>2793284639</td>
<td>MA</td>
</tr>
<tr>
<td>2723441590</td>
<td>AZ</td>
</tr>
<tr>
<td>2642551509</td>
<td>UT</td>
</tr>
</tbody>
</table>
```
Enabling View Access

You can query any view that you create. To enable other non-superusers to access a view, you must grant them USAGE privileges to the view schema, and SELECT privileges to the view itself. The following example grants user2 access to view schema1.view1:

```
=> GRANT USAGE ON schema schema1 TO user2;
=> GRANT SELECT ON schema1.view1 TO user2;
```

Important: If the view references an external table, you must also grant USAGE privileges to the external table's schema. So, if schema1.view1 references external table schema2.extTable1, you must also grant user2 USAGE privileges to schema2:

```
=> GRANT USAGE on schema schema2 to user2;
```

For more information see GRANT (View).

View Execution

When Vertica processes a query that contains a view, it treats the view as a subquery. Vertica executes the query by expanding it to include the query in the view definition. For example, Vertica expands the query on the view myview shown in CREATE VIEW Example, to include the query that the view encapsulates, as follows:

```
=> SELECT * FROM
   (SELECT SUM(annual_income), customer_state FROM public.customer_dimension
    WHERE customer_key IN
      (SELECT customer_key FROM store.store_sales_fact)
    GROUP BY customer_state
    ORDER BY customer_state ASC)
   AS ship where sum > 2000000000;
```

View Optimization

If you query a view and your query only includes columns from a subset of the tables that are joined in that view, Vertica executes that query by expanding it to include only those tables. This optimization requires one of the following conditions to be true:
- Join columns are foreign and primary keys.
- The join is a left or right outer join on columns with unique values.

**View Sort Order**

When processing a query on a view, Vertica considers the ORDER BY clause only in the outermost query. If the view definition includes an ORDER BY clause, Vertica ignores it. Thus, in order to sort the results returned by a view, you must specify the ORDER BY clause in the outermost query:

```sql
=> SELECT * FROM view-name ORDER BY view-column;
```

**Note:** One exception applies: Vertica sorts view data when the view includes a LIMIT clause. In this case, Vertica must sort the data before it can process the LIMIT clause.

For example, the following view definition contains an ORDER BY clause inside a FROM subquery:

```sql
=> CREATE VIEW myview AS SELECT SUM(annual_income), customer_state FROM public.customer_dimension
   WHERE customer_key IN
     (SELECT customer_key FROM store.store_sales_fact)
   GROUP BY customer_state
   ORDER BY customer_state ASC;
```

When you query the view, Vertica does not sort the data:

```sql
=> SELECT * FROM myview WHERE SUM > 200000000;
```

<table>
<thead>
<tr>
<th>SUM</th>
<th>customer_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>5225333668</td>
<td>MI</td>
</tr>
<tr>
<td>2832710696</td>
<td>TN</td>
</tr>
<tr>
<td>14215397659</td>
<td>TX</td>
</tr>
<tr>
<td>4907216137</td>
<td>CO</td>
</tr>
<tr>
<td>2793284639</td>
<td>MA</td>
</tr>
<tr>
<td>3769455689</td>
<td>CT</td>
</tr>
<tr>
<td>3310667307</td>
<td>IN</td>
</tr>
<tr>
<td>2723441590</td>
<td>AZ</td>
</tr>
<tr>
<td>2642551509</td>
<td>UT</td>
</tr>
<tr>
<td>3330524215</td>
<td>FL</td>
</tr>
<tr>
<td>2128169759</td>
<td>NV</td>
</tr>
<tr>
<td>29253817901</td>
<td>CA</td>
</tr>
<tr>
<td>4581840708</td>
<td>IL</td>
</tr>
<tr>
<td>2806150503</td>
<td>PA</td>
</tr>
</tbody>
</table>
```

(14 rows)

To return sorted results, the outer query must include an ORDER BY clause:
Run-Time Errors

If Vertica does not have to evaluate an expression that would generate a run-time error in order to answer a query, the run-time error might not occur.

For example, the following query returns an error, because TO_DATE cannot convert the string F to the specified date format:

```sql
=> SELECT TO_DATE('F','dd mm yyyy') FROM customer_dimension;
ERROR: Invalid input for DD: "F"
```

Now create a view using the same query:

```sql
=> CREATE VIEW temp AS SELECT TO_DATE('F','dd mm yyyy')
   FROM customer_dimension;
CREATE VIEW
```

In many cases, this view generates the same error message. For example:

```sql
=> SELECT * FROM temp;
ERROR: Invalid input for DD: "F"
```

However, if you query that view with the COUNT function, Vertica returns with the desired results:

```sql
=> SELECT COUNT(*) FROM temp;
COUNT
------
  100
(1 row)
```
This behavior works as intended. You can create views that contain subqueries, where not every row is intended to pass the predicate.

**Managing Views**

**Obtaining View Information**

You can query system tables `VIEWS` and `VIEW_COLUMNS` to obtain information about existing views—for example, a view's definition and the attributes of columns that comprise that view.

**Renaming a View**

Use `ALTER VIEW` to rename a view.

**Dropping a View**

Use `DROP VIEW` to drop a view. Only the specified view is dropped. Vertica does not support `CASCADE` functionality for views, and does not check for dependencies. Dropping a view causes any view that references it to fail.

**Disabling and Re-enabling Views**

If you drop a table that is referenced by a view, Vertica does not drop the view. However, attempts to use that view or access information about it from system table `VIEW_COLUMNS` return an error that the referenced table does not exist. If you restore that table, Vertica also re-enables usage of the view.
Flattened Tables

Highly normalized database design often uses a star or snowflake schema model, comprising multiple large fact tables and many smaller dimension tables. Queries typically involve joins between a large fact table and multiple dimension tables. Depending on the number of tables and quantity of data that are joined, these queries can incur significant overhead.

To avoid this problem, some users create wide tables that combine all fact and dimension table columns that their queries require. These tables can dramatically speed up query execution. However, maintaining redundant sets of normalized and denormalized data has its own administrative costs.

Denormalized, or flattened, tables, can minimize these problems. Flattened tables can include columns that get their values by querying other tables. Operations on the source tables and flattened table are decoupled; changes in one are not automatically propagated to the other. This minimizes the overhead that is otherwise typical of denormalized tables.

A flattened table defines derived columns with one of the following column constraint clauses:

- **DEFAULT** `query-expression` sets the column value on two events:
  - When the column is created with `CREATE TABLE` or `ALTER TABLE...ADD COLUMN`.
  - Any table load operation such as `INSERT`.

- **SET USING** `query-expression` sets the column value only when the function `REFRESH_COLUMNS` is invoked.

In both cases, `query-expression` must return only one row and column value, or none. If the query returns no rows, the column value is set to NULL.

Like other tables defined in Vertica, you can add and remove DEFAULT and SET USING columns from a flattened table at any time. Vertica enforces dependencies between a flattened table and the tables that it queries. For details, see Modifying SET USING and DEFAULT Columns.

Flattened Table Example

In the following example, columns `orderFact.cust_name` and `orderFact.cust_gender` are defined as SET USING and DEFAULT columns, respectively. Both columns obtain their values by querying table `custDim`:
CREATE TABLE public.custDim
  (cid int PRIMARY KEY NOT NULL,
   name varchar(20),
   age int,
   gender varchar(1)
  );
CREATE TABLE public.orderFact
  (order_id int PRIMARY KEY NOT NULL,
   cid int REFERENCES public.custDim(cid),
   cust_name varchar(20) SET USING (SELECT name FROM public.custDim WHERE (custDim.cid = orderFact.cid)),
   cust_gender varchar(1) DEFAULT (SELECT gender FROM public.custDim WHERE (custDim.cid = orderFact.cid)),
   amount numeric(12,2)
  );

The following INSERT commands load data into both tables:

INSERT INTO custDim VALUES(1, 'Alice', 25, 'F');
INSERT INTO custDim VALUES(2, 'Boz', 30, 'M');
INSERT INTO custDim VALUES(3, 'Eva', 32, 'F');
INSERT INTO orderFact (order_id, cid, amount) VALUES(100, 1, 15);
INSERT INTO orderFact (order_id, cid, amount) VALUES(200, 1, 1000);
INSERT INTO orderFact (order_id, cid, amount) VALUES(300, 2, -50);
INSERT INTO orderFact (order_id, cid, amount) VALUES(400, 3, 100);
INSERT INTO orderFact (order_id, cid, amount) VALUES(500, 2, 200);
COMMIT;

When you query the tables, Vertica returns the following result sets:

SELECT * FROM custDim;
<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
<th>age</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>25</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>Boz</td>
<td>30</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>Eva</td>
<td>32</td>
<td>F</td>
</tr>
</tbody>
</table>
(3 rows)

SELECT * FROM orderFact ORDER BY cid;
<table>
<thead>
<tr>
<th>order_id</th>
<th>cid</th>
<th>cust_name</th>
<th>cust_gender</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>F</td>
<td>15.00</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>F</td>
<td>1000.00</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>M</td>
<td>-50.00</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>M</td>
<td>200.00</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>3</td>
<td>F</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
(5 rows)

Vertica automatically populates the DEFAULT column orderFact.cust_gender, but the SET USING column orderFact.cust_name remains NULL. To populate this column, you must call the function REFRESH_COLUMNS on flattened table orderFact. This function invokes the SET USING query for column orderFact.cust_name and populates the column from the result set:
Creating Flattened Tables

A flattened table is typically a fact table where one or more columns query other tables for their values, through DEFAULT or SET USING constraints. Like other columns, you can set these constraints when you create the flattened table, or any time thereafter by modifying the table schema:

```
CREATE TABLE ... (...,
  column-name data-type { DEFAULT | SET USING } expression,...)
ALTER TABLE ... ADD COLUMN column-name { DEFAULT | SET USING } expression
ALTER TABLE ... ALTER COLUMN column-name { SET DEFAULT | SET USING } expression
```

DEFAULT and SET USING constraints can be used for columns of all data types. The expressions that you set for these constraints are stored in the system table COLUMNS, in columns COLUMN_DEFAULT and COLUMN_SET_USING. Both constraints support the same expressions. For detailed information, including restrictions, see Defining Column Values in the Administrator's Guide.

Supported Expressions

DEFAULT and SET USING generally support the same expressions. These include:

- Queries (see Flattened Tables)
- Other columns in the same table
- Literals (constants)
All operators supported by Vertica

The following categories of functions:

- Null-handling
- User-defined scalar
- System information
- String
- Mathematical
- Formatting

Disambiguating Predicate Columns

If a SET USING or DEFAULT query expression joins two columns with the same name, the column names must include their table names. Otherwise, Vertica assumes that both columns reference the dimension table, and the predicate always evaluates to true.

For example, tables `orderFact` and `custDim` both include column `cid`. Flattened table `orderFact` defines column `cust_name` with a SET USING query expression. Because the query predicate references columns `cid` from both tables, the column names are fully qualified:

```sql
CREATE TABLE public.orderFact
(
    ... cid int REFERENCES public.custDim(cid),
    cust_name varchar(20) SET USING (SELECT name FROM public.custDim WHERE (custDIM.cid = orderFact.cid)),
    ...)
```

Requirements and Restrictions

Required Privileges

Flattened tables require the following privileges:
To view flattened table data: READ privileges. No privileges are required on queried tables.

To add SET USING or DEFAULT columns that query other tables: SELECT privileges on the queried tables, CREATE privileges on the flattened table.

To load data: SELECT privileges on queried tables.

Running `REFRESH_COLUMNS` requires the following privileges on the flattened table:

- MODIFY privilege on the table, USAGE privilege on its schema
- For each SET USING column that queries another table or view: SELECT privilege on the queried table, USAGE privilege on its schema.

Restrictions

If you call `REFRESH_COLUMNS` on a SET USING column and specify the refresh mode as REBUILD, Vertica returns an error if any of the following conditions is true for that column:

- Specified as a table partition key.
- Included in a live aggregate projection or projection with expressions.
- Included in a projection's sort order or segmentation.
- Included in a projection, and the projection omits an anchor table column that is referenced in the column's SET USING expression.
- Included in a projection's GROUPED clause.

**SET USING versus DEFAULT**

Columns in a flattened table can query other tables with constraints SET USING and DEFAULT. In both cases, changes in the queried tables are not automatically propagated to the flattened table. The two constraints differ as described below.
DEFAULT Columns

Vertica executes DEFAULT queries only on new rows when they are added to the flattened table, through load operations such as INSERT and COPY. Thereafter, changes in the original data sources have no effect on the flattened table.

SET USING Columns

Vertica executes SET USING queries only when you invoke the function REFRESH_COLUMNS. Load operations set SET USING columns in new rows to NULL. After the load, you must call REFRESH_COLUMNS to populate these columns from the queried tables. This can be useful in two ways: you can defer the overhead of updating the flattened table to any time that is convenient; and you can repeatedly query source tables for new data.

One exception applies: when you use ALTER TABLE...ALTER COLUMN to apply SET USING to an existing column, or modify an existing SET USING expression. In this case, the DDL operation automatically invokes REFRESH_COLUMNS on the column, using UPDATE mode. After the refresh operation is complete, the DDL operation auto-commits the updates and returns. If the refresh operation fails, Vertica rolls back the entire DDL operation. Execution time can be significant if the refresh operation involves a large data set.

Tip: It might be more efficient to drop the column and add a new one. ALTERTABLE...ADD COLUMN does not call REFRESH_COLUMNS when you add a SET USING column to a table.

SET USING is especially useful for large flattened tables that reference data from multiple dimension tables. Often, only a small subset of SET USING columns are subject to change, and queries on the flattened table do not always require up-to-the-minute data. Given this scenario, you can refresh table content at regular intervals, or only during off-peak hours. One or both of these strategies can minimize maintenance costs, and facilitate performance when querying large data sets.

Combining DEFAULT and SET USING Constraints

A column can specify both DEFAULT and SET USING constraints, as follows:

column-name data-type DEFAULT default-expr SET USING using-expr

Typically, both constraints specify the same expression. In this case, you can define the column as follows:
column-name data-type DEFAULT USING expression

DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same restrictions.

Example

The following SQL illustrates differences between SET USING and DEFAULT constraints. The examples use the custDim and orderFact tables described in Flattened Table Example.

The following UPDATE statement updates the custDim table:

```sql
=> UPDATE custDim SET name='Roz', gender='F' WHERE cid=2;
OUTPUT
-------
 1
(1 row)
=> COMMIT;
COMMIT
```

Changes are not propagated to flattened table orderFact, which includes SET USING and DEFAULT columns cust_name and cust_gender, respectively:

```sql
=> SELECT * FROM custDim ORDER BY cid;
cid | name  | age | gender
----------
 1  | Alice | 25  | F
 2  | Roz   | 30  | F
 3  | Eva   | 32  | F
(3 rows)
=> SELECT * FROM orderFact ORDER BY cid;
order_id | cid | cust_name | cust_gender | amount
----------
 100 | 1  | Alice     | F           | 15.00
 200 | 1  | Alice     | F           | 1000.00
 300 | 2  | Boz       | M           | -50.00
 500 | 2  | Boz       | M           | 200.00
 400 | 3  | Eva       | F           | 100.00
(5 rows)
```

The following INSERT statement invokes the cust_gender column's DEFAULT query and sets that column to F. The load operation does not invoke the cust_name column's SET USING query, so cust_name is set to null:

```sql
=> INSERT INTO orderFact(order_id, cid, amount) VALUES(500, 3, 750);
OUTPUT
-------
 1
(1 row)
```
To set a value in cust_name, invoke its SET USING query by calling REFRESH_COLUMNS:

```sql
=> SELECT REFRESH_COLUMNS ('orderFact', '');

<table>
<thead>
<tr>
<th>REFRESH_COLUMNS</th>
<th>refresh_columns completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>

=> COMMIT;

=> SELECT * FROM orderFact ORDER BY cid;

<table>
<thead>
<tr>
<th>order_id</th>
<th>cid</th>
<th>cust_name</th>
<th>cust_gender</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>Alice</td>
<td>F</td>
<td>15.00</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>Alice</td>
<td>F</td>
<td>1000.00</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>Boz</td>
<td>M</td>
<td>-50.00</td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>Boz</td>
<td>M</td>
<td>200.00</td>
</tr>
<tr>
<td>400</td>
<td>3</td>
<td>Eva</td>
<td>F</td>
<td>100.00</td>
</tr>
<tr>
<td>500</td>
<td>3</td>
<td>Eva</td>
<td>F</td>
<td>750.00</td>
</tr>
</tbody>
</table>

(6 rows)
```

REFRESH_COLUMNS executes cust_name's SET USING query: it queries the name column in table custDim and updates cust_name with the following values:

- Sets cust_name in the new row to Eva.
- Returns updated values for cid=2, and changes Boz to Roz.

REFRESH_COLUMNS only affects the values in column cust_name. Values in column gender are unchanged, so settings for rows where cid=2 (Roz) remain set to M. To repopulate orderFact.cust_gender with default values from custDim.gender, call UPDATE on orderFact:

```sql
=> UPDATE orderFact SET cust_gender=DEFAULT WHERE cust_name='Roz';

<table>
<thead>
<tr>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

=> COMMIT;

=> COMMIT
```
Modifying SET USING and DEFAULT Columns

Removing SET USING and DEFAULT Constraints

You remove a column's SET USING or DEFAULT constraint with `ALTER TABLE...ALTER COLUMN`, as follows:

```
ALTER TABLE table-name ALTER COLUMN column-name DROP { SET USING | DEFAULT };
```

Vertica removes the constraint from the specified column, but the column and its data are otherwise unaffected.

Modifying a SET USING and DEFAULT Expression

You can change the query expression of an existing SET USING or DEFAULT column by calling `ALTER TABLE...ALTER COLUMN`. When you modify a SET USING column expression, Vertica automatically invokes `REFRESH_COLUMNS` using UPDATE mode, and repopulates that column. If a large number of rows must be refreshed, the following approach might be preferable:

1. Drop the column with `ALTER TABLE...DROP COLUMN`.

2. Call `ALTER TABLE...ADD COLUMN` to add a new one with the desired SET USING expression.

This approach has two benefits:

- Avoids the immediate call to `REFRESH_COLUMNS`, and defers the overhead that is otherwise incurred.
- Lets you specify REBUILD mode when you call REFRESH_COLUMNS, which in many cases repopulates the column more efficiently.

## Dropping Columns Queried by SET USING or DEFAULT

Vertica enforces dependencies between a flattened table and the tables that it queries. Attempts to drop a queried column or its table return an error unless the drop operation, such as DROP TABLE, also includes CASCADE. Vertica implements CASCADE by removing the SET USING or DEFAULT constraint from the flattened table. The table column and its data are otherwise unaffected.

For example, attempts to drop column name in table custDim returns a rollback error, as this column is referenced by SET USING column cust_name in orderFact:

```
=> ALTER TABLE custDim DROP COLUMN name;
ROLLBACK 7302: Cannot drop column "name" since it is referenced in the set using expression of table "public.orderFact", column "cust_name"
```

To drop this column, use the CASCADE option:

```
=> ALTER TABLE custDim DROP COLUMN name CASCADE;
=> ALTER TABLE public.orderFact ADD CONSTRAINT C_PRIMARY PRIMARY KEY (order_id) ENABLED;
=> ALTER TABLE public.orderFact ADD CONSTRAINT C_FOREIGN FOREIGN KEY (cid) references public.custDim (cid);
(1 row)
```

Vertica removes the SET USING constraint from orderFact.cust_name as part of the drop operation. However, cust_name retains the data that it derived from dropped column custDim.name:

```
=> SELECT EXPORT_TABLES('','orderFact');

---------------------------------------------
CREATE TABLE public.orderFact
(
  order_id int NOT NULL,
  cid int,
  cust_name varchar(20)
) distribute on (custDim.cid) primary key (order_id);

=> SELECT * FROM orderFact;
order_id | cid | cust_name | cust_gender | amount
---------+-------+-----------+-------------+----------
```
### Impact of SET USING Columns on License Limits

Vertica does not count the data in denormalized columns towards your raw data license limit. SET USING columns obtain their data by querying columns in other tables. You cannot modify SET USING data directly—for example, through DML operations such as INSERT and COPY. Thus, only the source data counts against your raw data license limit.

For a list of SET USING restrictions, see [Defining Column Values](#).
SQL Analytics

Vertica analytics are SQL functions based on the ANSI 99 standard. These functions handle complex analysis and reporting tasks—for example:

- Rank the longest-standing customers in a particular state.
- Calculate the moving average of retail volume over a specified time.
- Find the highest score among all students in the same grade.
- Compare the current sales bonus that salespersons received against their previous bonus.

Analytic functions return aggregate results but they do not group the result set. They return the group value multiple times, once per record. You can sort group values, or partitions, using a window ORDER BY clause, but the order affects only the function result set, not the entire query result set.

For details about supported functions, see Analytic Functions in the SQL Reference Manual.

Invoking Analytic Functions

You invoke analytic functions as follows:

```
analytic-function (arguments) OVER(
  [ window-partition-clause ]
  [ window-order-clause [ window-frame-clause ] ]
)
```

An analytic function's OVER clause can contain up to three sub-clauses, which specify how to partition and sort function input, and how to frame input with respect to the current row. Function input is the result set that the query returns after it evaluates FROM, WHERE, GROUP BY, and HAVING clauses.

Each function has its own OVER clause requirements. For example, some analytic functions do not support window order and window frame clauses.

Function Execution

An analytic function executes as follows:
1. Takes the input rows that the query returns after it performs all joins, and evaluates FROM, WHERE, GROUP BY, and HAVING clauses.

2. Groups input rows according to the window partition (PARTITION BY) clause. If this clause is omitted, all input rows are treated as a single partition.

3. Sorts rows within each partition according to window order (ORDER BY) clause.

4. If the OVER clause includes a window order clause, the function checks for a window frame clause and executes it as it processes each input row. If the OVER clause omits a window frame clause, the function treats the entire partition as a window frame.

Restrictions

- Analytic functions are allowed only in a query's SELECT and ORDER BY clauses.
- Analytic functions cannot be nested. For example, the following query throws an error:

  ```sql
  => SELECT MEDIAN(RANK() OVER(ORDER BY sal) OVER())
  ```

Analytic Functions Versus Aggregate Functions

Like aggregate functions, analytic functions return aggregate results, but analytics do not group the result set. Instead, they return the group value multiple times with each record, allowing further analysis.

Analytic queries generally run faster and use fewer resources than aggregate queries.

<table>
<thead>
<tr>
<th>Aggregate functions</th>
<th>Analytic functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return a single summary value.</td>
<td>Return the same number of rows as the input.</td>
</tr>
<tr>
<td>Define the groups of rows on which they operate through the SQL GROUP BY clause.</td>
<td>Define the groups of rows on which they operate through window partition and window frame clauses.</td>
</tr>
</tbody>
</table>
Examples

The examples below contrast the aggregate function \texttt{COUNT} with its analytic counterpart \texttt{COUNT}. The examples use the \texttt{employees} table as defined below:

```
CREATE TABLE employees(emp_no INT, dept_no INT);
INSERT INTO employees VALUES(1, 10);
INSERT INTO employees VALUES(2, 30);
INSERT INTO employees VALUES(3, 30);
INSERT INTO employees VALUES(4, 10);
INSERT INTO employees VALUES(5, 30);
INSERT INTO employees VALUES(6, 20);
INSERT INTO employees VALUES(7, 20);
INSERT INTO employees VALUES(8, 20);
INSERT INTO employees VALUES(9, 20);
INSERT INTO employees VALUES(10, 20);
INSERT INTO employees VALUES(11, 20);
COMMIT;
```

When you query this table, the following result set returns:

```
=> SELECT * FROM employees ORDER BY emp_no;
  emp_no | dept_no
----------+-------
     1    |   10
     2    |   30
     3    |   30
     4    |   10
     5    |   30
     6    |   20
     7    |   20
     8    |   20
     9    |   20
    10    |   20
     11   |   20
(11 rows)
```

Below, two queries use the \texttt{COUNT} function to count the number of employees in each department. The query on the left uses aggregate function \texttt{COUNT}; the query on the right uses analytic function \texttt{COUNT}:

<table>
<thead>
<tr>
<th>Aggregate COUNT</th>
<th>Analytics COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>=&gt; SELECT dept_no, COUNT(*) AS emp_count FROM employees GROUP BY dept_no ORDER BY dept_no;</td>
<td>=&gt; SELECT emp_no, dept_no, COUNT(*) OVER(PARTITION BY dept_no ORDER BY emp_no) AS emp_count FROM employees;</td>
</tr>
<tr>
<td>dept_no</td>
<td>emp_count</td>
</tr>
<tr>
<td>---------+-----------</td>
<td></td>
</tr>
</tbody>
</table>
Aggregate function `COUNT` returns one row per department for the number of employees in that department.

Within each `dept_no` partition analytic function `COUNT` returns a cumulative count of employees. The count is ordered by `emp_no`, as specified by the `ORDER BY` clause.

See Also

- Analytic Query Examples

Window Partitioning

Optionally specified in an analytic function's `OVER` clause, a partition (`PARTITION BY`) clause groups input rows before the function processes them. Window partitioning is similar to an aggregate function's `GROUP BY` clause, except it returns exactly one result row per input row. If you omit the window partition clause, the function treats all input rows as a single partition.

Specifying Window Partitioning

You specify window partitioning in the analytic function's `OVER` clause, as shown below. The window partition clause is set off in bold:

```sql
OVER( { PARTITION BY expression  | PARTITION BEST | PARTITION NODES }
      window-order-clause
      window-frame-clause )
```

For syntax details, see Window Partition Clause in the SQL Reference Manual.
Examples

The examples in this topic use the allsales schema defined in Invoking Analytic Functions.

```sql
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

**Median of sales within each state**

The following query uses the analytic `window-partition-clause` to calculate the median of sales within each state. The analytic function is computed per partition and starts over again at the beginning of the next partition.

```sql
=> SELECT state, name, sales, MEDIAN(sales) OVER (PARTITION BY state) AS median from allsales;
```

Results are grouped into partitions for MA (35) and NY (20) under the median column.

<table>
<thead>
<tr>
<th>state</th>
<th>name</th>
<th>sales</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>C</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>NY</td>
<td>B</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>NY</td>
<td>F</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>MA</td>
<td>G</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>MA</td>
<td>D</td>
<td>20</td>
<td>35</td>
</tr>
<tr>
<td>MA</td>
<td>E</td>
<td>50</td>
<td>35</td>
</tr>
<tr>
<td>MA</td>
<td>A</td>
<td>60</td>
<td>35</td>
</tr>
</tbody>
</table>

(7 rows)

**Median of sales among all states**

The following query calculates the median of total sales among states. When you use `OVER()` with no parameters, there is one partition—the entire input:

```sql
=> SELECT state, sum(sales), median(SUM(sales)) OVER () AS median from allsales GROUP BY state;
```

<table>
<thead>
<tr>
<th>state</th>
<th>sum</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>75</td>
<td>107.5</td>
</tr>
<tr>
<td>MA</td>
<td>140</td>
<td>107.5</td>
</tr>
</tbody>
</table>

(2 rows)

**Sales larger than median (evaluation order)**

Analytic functions are evaluated after all other clauses except the query's final SQL ORDER BY
clause. So a query that asks for all rows with sales larger than the median returns an error because the WHERE clause is applied before the analytic function and column m does not yet exist:

```sql
=> SELECT name, sales, MEDIAN(sales) OVER () AS m
   FROM allsales WHERE sales > m;
ERROR 2624: Column "m" does not exist
```

You can work around this by placing in a subquery the predicate WHERE sales > m:

```sql
=> SELECT * FROM
   (SELECT name, sales, MEDIAN(sales) OVER () AS m FROM allsales) sq
   WHERE sales > m;
```

<table>
<thead>
<tr>
<th>name</th>
<th>sales</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>A</td>
<td>60</td>
<td>20</td>
</tr>
</tbody>
</table>
(3 rows)

For more examples, see Analytic Query Examples.

## Window Ordering

Window ordering specifies how to sort rows that are supplied to the analytic function. You specify window ordering through an ORDER BY clause in the function's OVER clause, as shown below. If the OVER clause includes a window partition clause, rows are sorted within each partition. An window order clause also creates a default window frame if none is explicitly specified.

The window order clause only specifies order within a window result set. The query can have its own ORDER BY clause outside the OVER clause. This has precedence over the window order clause and orders the final result set.

### Specifying Window Order

You specify a window frame in the analytic function's OVER clause, as shown below. The window order clause is set off in bold:

```sql
OVER(
    [ window-partition-clause ]
    ORDER BY { expression[ sort-qualifiers ] } [, ...]
    [ window-frame-clause ]
)
```

`sort-qualifiers =`
For syntax details, see Window Order Cause in the SQL Reference Manual.

Analytic Function Usage

Analytic aggregation functions such as SUM support window order clauses.

Required Usage
The following functions require a window order clause:

- RANK
- DENSE_RANK
- LEAD
- LAG
- PERCENT_RANK
- CUME_DIST
- NTILE

Invalid Usage
You cannot use a window order clause with the following functions:

- PERCENTILE_CONT
- PERCENTILE_DISC
- MEDIAN

Examples

The examples below use the allsales table schema:

```sql
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
```
Example 1

The following query orders sales inside each state partition:

```sql
=> SELECT state, sales, name, RANK() OVER(PARTITION BY state ORDER BY sales) AS RANK
FROM allsales;
```

<table>
<thead>
<tr>
<th>state</th>
<th>sales</th>
<th>name</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>10</td>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>MA</td>
<td>20</td>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>MA</td>
<td>50</td>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>MA</td>
<td>60</td>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>NY</td>
<td>15</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>NY</td>
<td>20</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td>F</td>
<td>3</td>
</tr>
</tbody>
</table>

(7 rows)

Example 2

The following query's final ORDER BY clause sorts results by name:

```sql
=> SELECT state, sales, name, RANK() OVER(PARTITION BY state ORDER BY sales) AS RANK
FROM allsales ORDER BY name;
```

<table>
<thead>
<tr>
<th>state</th>
<th>sales</th>
<th>name</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>60</td>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>NY</td>
<td>20</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>NY</td>
<td>15</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>MA</td>
<td>20</td>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>MA</td>
<td>50</td>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>MA</td>
<td>10</td>
<td>G</td>
<td>1</td>
</tr>
</tbody>
</table>

(7 rows)
Window Framing

The window frame of an analytic function comprises a set of rows relative to the row that is currently being evaluated by the function. After the analytic function processes that row and its window frame, Vertica advances the current row and adjusts the frame boundaries accordingly. If the OVER clause also specifies a partition, Vertica also checks that frame boundaries do not cross partition boundaries. This process repeats until the function evaluates the last row of the last partition.

Specifying a Window Frame

You specify a window frame in the analytic function's OVER clause, as shown below. The window frame clause is set off in bold:

```
OVER(
   [ window-partition-clause]
   window-order-clause
   { ROWS | RANGE } { BETWEEN start-point AND end-point } | start-point
)
```

`start-point` and `end-point` specify the window frame's offset from the current row, as follows:

```
{ UNBOUNDED {PRECEDING | FOLLOWING} |
  CURRENT ROW |
  constant-value {PRECEDING | FOLLOWING}
}
```

The keywords `ROWS` and `RANGE` specify whether the offset is physical or logical. If you specify only a start point, Vertica creates a window from that point to the current row.

For syntax details, see `window-frame-clause` in the SQL Reference Manual.

Requirements

In order to specify a window frame, the OVER must also specify a `window order (ORDER BY)` clause. If the OVER clause includes a window order clause but omits specifying a window frame, the function creates a default frame that extends from the first row in the current partition to the current row. This is equivalent to the following clause:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```
Window Aggregate Functions

Analytic functions that support window frames are called window aggregates. They return information such as moving averages and cumulative results. To use the following functions as window (analytic) aggregates, instead of basic aggregates, the OVER clause must specify a window order clause and, optionally, a window frame clause. If the OVER clause omits specifying a window frame, the function creates a default window frame as described earlier.

The following analytic functions support window frames:

- AVG
- COUNT
- MAX / MIN
- SUM
- STDDEV / STDDEV_POP / STDDEV_SAMP
- VARIANCE / VAR_POP / VAR_SAMP

Note: Functions FIRST_VALUE and LAST_VALUE also support window frames, but they are only analytic functions with no aggregate counterpart. EXPONENTIAL MOVING AVERAGE, LAG, and LEAD analytic functions do not support window frames.

A window aggregate with an empty OVER clause creates no window frame. The function is used as a reporting function, where all input is treated as a single partition.

Windows with a Physical Offset (ROWS)

The keyword ROWS in a window frame clause specifies window dimensions as the number of rows relative to the current row. The value can be INTEGER data type only.

Note: The value returned by an analytic function with a physical offset is liable to produce nondeterministic results unless the ordering expression results in a unique ordering. To achieve unique ordering, the window order clause might need to specify multiple columns.
Examples

The examples on this page use the emp table schema:

```sql
CREATE TABLE emp(deptno INT, sal INT, empno INT);
INSERT INTO emp VALUES(10,101,1);
INSERT INTO emp VALUES(10,104,4);
INSERT INTO emp VALUES(20,100,11);
INSERT INTO emp VALUES(20,109,7);
INSERT INTO emp VALUES(20,109,6);
INSERT INTO emp VALUES(20,109,8);
INSERT INTO emp VALUES(20,110,10);
INSERT INTO emp VALUES(20,110,9);
INSERT INTO emp VALUES(30,102,2);
INSERT INTO emp VALUES(30,103,3);
INSERT INTO emp VALUES(30,105,5);
COMMIT;
```

The following query invokes COUNT to count the current row and the rows preceding it, up to two rows:

```sql
SELECT deptno, sal, empno, COUNT(*) OVER 
(PARTITION BY deptno ORDER BY sal ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) 
AS count FROM emp;
```

The OVER clause contains three components:

- **Window partition clause** PARTITION BY deptno
- **Order by clause** ORDER BY sal
- **Window frame clause** ROWS BETWEEN 2 PRECEDING AND CURRENT ROW. This clause defines window dimensions as extending from the current row through the two rows that precede it.

The query returns results that are divided into three partitions, indicated below as red lines. Within the second partition (deptno=20), COUNT processes the window frame clause as follows:

1. Creates the first window (green box). This window comprises a single row, as the current row (blue box) is also the the partition's first row. Thus, the value in the count column shows the number of rows in the current window, which is 1:
2. After COUNT processes the partition's first row, it resets the current row to the partition's second row. The window now spans the current row and the row above it, so COUNT returns a value of 2:

```
<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
```

3. After COUNT processes the partition's second row, it resets the current row to the partition's third row. The window now spans the current row and the two rows above it, so
COUNT returns a value of 3:

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

4. Thereafter, COUNT continues to process the remaining partition rows and moves the window accordingly, but the window dimensions (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) remain unchanged as three rows. Accordingly, the value in the count column also remains unchanged (3):

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Windows with a Logical Offset (RANGE)

The RANGE keyword defines an analytic window frame as a logical offset from the current row.

**Note:** The value returned by an analytic function with a logical offset is always deterministic.

For each row, an analytic function uses the window order clause (ORDER BY) column or expression to calculate window frame dimensions as follows:
1. Within the current partition, evaluates the ORDER_BY value of the current row against the ORDER_BY values of contiguous rows.

2. Determines which of these rows satisfy the specified range requirements relative to the current row.

3. Creates a window frame that includes only those rows.

4. Executes on the current window.

Example

This example uses the table property_sales, which contains data about neighborhood home sales:

```
=> SELECT property_key, neighborhood, sell_price FROM property_sales ORDER BY neighborhood, sell_price;

<table>
<thead>
<tr>
<th>property_key</th>
<th>neighborhood</th>
<th>sell_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10918</td>
<td>Jamaica Plain</td>
<td>353000</td>
</tr>
<tr>
<td>10921</td>
<td>Jamaica Plain</td>
<td>450000</td>
</tr>
<tr>
<td>10927</td>
<td>Jamaica Plain</td>
<td>450000</td>
</tr>
<tr>
<td>10922</td>
<td>Jamaica Plain</td>
<td>474000</td>
</tr>
<tr>
<td>10919</td>
<td>Jamaica Plain</td>
<td>515000</td>
</tr>
<tr>
<td>10917</td>
<td>Jamaica Plain</td>
<td>675000</td>
</tr>
<tr>
<td>10924</td>
<td>Jamaica Plain</td>
<td>675000</td>
</tr>
<tr>
<td>10920</td>
<td>Jamaica Plain</td>
<td>705000</td>
</tr>
<tr>
<td>10923</td>
<td>Jamaica Plain</td>
<td>710000</td>
</tr>
<tr>
<td>10926</td>
<td>Jamaica Plain</td>
<td>875000</td>
</tr>
<tr>
<td>10925</td>
<td>Jamaica Plain</td>
<td>900000</td>
</tr>
<tr>
<td>10930</td>
<td>Roslindale</td>
<td>300000</td>
</tr>
<tr>
<td>10928</td>
<td>Roslindale</td>
<td>422000</td>
</tr>
<tr>
<td>10932</td>
<td>Roslindale</td>
<td>450000</td>
</tr>
<tr>
<td>10929</td>
<td>Roslindale</td>
<td>485000</td>
</tr>
<tr>
<td>10931</td>
<td>Roslindale</td>
<td>519000</td>
</tr>
<tr>
<td>10938</td>
<td>West Roxbury</td>
<td>479000</td>
</tr>
<tr>
<td>10933</td>
<td>West Roxbury</td>
<td>550000</td>
</tr>
<tr>
<td>10937</td>
<td>West Roxbury</td>
<td>550000</td>
</tr>
<tr>
<td>10934</td>
<td>West Roxbury</td>
<td>574000</td>
</tr>
<tr>
<td>10935</td>
<td>West Roxbury</td>
<td>598000</td>
</tr>
<tr>
<td>10936</td>
<td>West Roxbury</td>
<td>615000</td>
</tr>
<tr>
<td>10939</td>
<td>West Roxbury</td>
<td>720000</td>
</tr>
</tbody>
</table>
```

(23 rows)

The analytic function \texttt{AVG} can obtain the average of proximate selling prices within each neighborhood. The following query calculates for each home the average sale for all other neighborhood homes whose selling price was $50k higher or lower:

```
=> SELECT property_key, neighborhood, sell_price, AVG(sell_price) OVER(PARTITION BY neighborhood ORDER BY sell_price RANGE BETWEEN 50000 PRECEDING and 50000 FOLLOWING)::int AS comp_sales
```
AVG processes this query as follows:

1. AVG evaluates row 1 of the first partition (Jamaica Plain), but finds no sales within $50k of this row's sell_price, ($353k). AVG creates a window that includes this row only, and returns an average of 353k for row 1:

   ![Diagram of AVG on row 1]

2. AVG evaluates row 2 and finds three sell_price values within $50k of the current row. AVG creates a window that includes these three rows, and returns an average of 458k for
row 2:

<table>
<thead>
<tr>
<th>property_key</th>
<th>neighborhood</th>
<th>sell_price</th>
<th>comp_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>10918</td>
<td>Jamaica Plain</td>
<td>353000</td>
<td>353000</td>
</tr>
<tr>
<td>10927</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10921</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10922</td>
<td>Jamaica Plain</td>
<td>474000</td>
<td>472250</td>
</tr>
</tbody>
</table>

3. AVG evaluates row 3 and finds the same three sell_price values within $50k of the current row. AVG creates a window identical to the one before, and returns the same average of 458k for row 3:

<table>
<thead>
<tr>
<th>property_key</th>
<th>neighborhood</th>
<th>sell_price</th>
<th>comp_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>10918</td>
<td>Jamaica Plain</td>
<td>353000</td>
<td>353000</td>
</tr>
<tr>
<td>10927</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10921</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10922</td>
<td>Jamaica Plain</td>
<td>474000</td>
<td>472250</td>
</tr>
</tbody>
</table>

4. AVG evaluates row 4 and finds four sell_price values within $50k of the current row. AVG expands its window to include rows 2 through 5, and returns an average of $472.25k for row 4:

<table>
<thead>
<tr>
<th>property_key</th>
<th>neighborhood</th>
<th>sell_price</th>
<th>comp_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>10918</td>
<td>Jamaica Plain</td>
<td>353000</td>
<td>353000</td>
</tr>
<tr>
<td>10927</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10921</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10922</td>
<td>Jamaica Plain</td>
<td>474000</td>
<td>472250</td>
</tr>
<tr>
<td>10919</td>
<td>Jamaica Plain</td>
<td>515000</td>
<td>494500</td>
</tr>
<tr>
<td>10917</td>
<td>Jamaica Plain</td>
<td>675000</td>
<td>691250</td>
</tr>
</tbody>
</table>
5. In similar fashion, AVG evaluates the remaining rows in this partition. When the function evaluates the first row of the second partition (Roslindale), it resets the window as follows:

<table>
<thead>
<tr>
<th>property_key</th>
<th>neighborhood</th>
<th>sell_price</th>
<th>comp_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>10918</td>
<td>Jamaica Plain</td>
<td>353000</td>
<td>353000</td>
</tr>
<tr>
<td>10927</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10921</td>
<td>Jamaica Plain</td>
<td>450000</td>
<td>458000</td>
</tr>
<tr>
<td>10922</td>
<td>Jamaica Plain</td>
<td>474000</td>
<td>472250</td>
</tr>
<tr>
<td>10919</td>
<td>Jamaica Plain</td>
<td>515000</td>
<td>494500</td>
</tr>
<tr>
<td>10917</td>
<td>Jamaica Plain</td>
<td>675000</td>
<td>691250</td>
</tr>
<tr>
<td>10924</td>
<td>Jamaica Plain</td>
<td>675000</td>
<td>691250</td>
</tr>
<tr>
<td>10920</td>
<td>Jamaica Plain</td>
<td>705000</td>
<td>691250</td>
</tr>
<tr>
<td>10923</td>
<td>Jamaica Plain</td>
<td>710000</td>
<td>691250</td>
</tr>
<tr>
<td>10926</td>
<td>Jamaica Plain</td>
<td>875000</td>
<td>887500</td>
</tr>
<tr>
<td>10925</td>
<td>Jamaica Plain</td>
<td>900000</td>
<td>887500</td>
</tr>
<tr>
<td>10930</td>
<td>Roslindale</td>
<td>300000</td>
<td>300000</td>
</tr>
<tr>
<td>10928</td>
<td>Roslindale</td>
<td>422000</td>
<td>436000</td>
</tr>
</tbody>
</table>

Restrictions

If RANGE specifies a constant value, that value's data type and the window's ORDER BY data type must be the same. The following exceptions apply:

- RANGE can specify INTERVAL Year to Month if the window order clause data type is one of following: TIMESTAMP, TIMESTAMP WITH TIMEZONE, or DATE. TIME and TIME WITH TIMEZONE are not supported.

- RANGE can specify INTERVAL Day to Second if the window order clause data is one of following: TIMESTAMP, TIMESTAMP WITH TIMEZONE, DATE, TIME, or TIME WITH TIMEZONE.

The window order clause must specify one of the following data types: NUMERIC, DATE/TIME, FLOAT or INTEGER. This requirement is ignored if the window specifies one of following frames:

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
- RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
Reporting Aggregates

Some of the analytic functions that take the window-frame-clause are the reporting aggregates. These functions let you compare a partition's aggregate values with detail rows, taking the place of correlated subqueries or joins.

- **AVG()**
- **COUNT()**
- **MAX()** and **MIN()**
- **SUM()**
- **STDDEV(), STDDEV_POP(),** and **STDDEV_SAMP()**
- **VARIANCE(), VAR_POP(),** and **VAR_SAMP()**

If you use a window aggregate with an empty OVER() clause, the analytic function is used as a reporting function, where the entire input is treated as a single partition.

About Standard Deviation and Variance Functions

With standard deviation functions, a low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data points are spread out over a large range of values.

Standard deviation is often graphed and a distributed standard deviation creates the classic bell curve.

Variance functions measure how far a set of numbers is spread out.

Examples

Think of the window for reporting aggregates as a window defined as UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING. The omission of a window-order-clause makes all rows in the partition also the window (reporting aggregates).

```sql
=> SELECT deptno, sal, empno, COUNT(sal)
    OVER (PARTITION BY deptno) AS count
FROM emp;
```

depthno | sal | empno | count
---------+-----+-------+-----
---------+-----+-------+-----
If the OVER() clause in the above query contained a window-order-clause (for example, ORDER BY sal), it would become a moving window (window aggregate) query with a default window of RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW:

```sql
=> SELECT deptno, sal, empno, COUNT(sal) OVER (PARTITION BY deptno ORDER BY sal) AS COUNT FROM emp;
```

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

(11 rows)

What About LAST_VALUE?

You might wonder why you couldn't just use the LAST_VALUE() analytic function.

For example, for each employee, get the highest salary in the department:

```sql
=> SELECT deptno, sal, empno, LAST_VALUE(empno) OVER (PARTITION BY deptno ORDER BY sal) AS lv FROM emp;
```

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>lv</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

(11 rows)
Due to default window semantics, LAST_VALUE does not always return the last value of a partition. If you omit the window-frame-clause from the analytic clause, LAST_VALUE operates on this default window. Results, therefore, can seem non-intuitive because the function does not return the bottom of the current partition. It returns the bottom of the window, which continues to change along with the current input row being processed.

Remember the default window:

```
OVER (PARTITION BY deptno ORDER BY sal)
```

is the same as:

```
OVER(PARTITION BY deptno ORDER BY salROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```
If you want to return the last value of a partition, use UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

```sql
=> SELECT deptno, sal, empno, LAST_VALUE(empno) OVER (PARTITION BY deptno ORDER BY sal ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM emp;
```
Vertica recommends that you use LAST_VALUE with the window-order-clause to produce deterministic results.

In the following example, empno 6, 7, and 8 have the same salary, so they are in adjacent rows. empno 8 appears first in this case but the order is not guaranteed.

```
deptno | sal | empno | lv
-------+-----+-------+----
  10 | 101 |   1 |  4
  10 | 104 |   4 |  4
  20 | 100 |  11 |  9
  20 | 109 |   7 |  9
  20 | 109 |   6 |  9
  20 | 109 |   3 |  9
  20 | 110 |  10 |  9
  20 | 110 |   9 |  9
  30 | 102 |   2 |  5
  30 | 103 |   3 |  5
  30 | 105 |   5 |  5
```

Notice in the output above, the last value is 7, which is the last row from the partition deptno = 20. If the rows have a different order, then the function returns a different value:
Now the last value is 6, which is the last row from the partition deptno = 20. The solution is to add a unique key to the sort order. Even if the order of the query changes, the result will always be the same, and so deterministic.

=> SELECT deptno, sal, empno, LAST_VALUE(empno)
OVER (PARTITION BY deptno ORDER BY sal, empno
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) as lv
FROM emp;

deptno | sal | empno | lv
--------+-----+-------+-----
 10     | 101 | 1     | 4
 10     | 104 | 4     | 4

20     | 100 | 11    | 6

20     | 109 | 109   | 6

20     | 109 | 109   | 8

30     | 102 | 2     | 5

30     | 103 | 3     | 5

30     | 105 | 5     | 5

Notice how the rows are now ordered by empno, the last value stays at 8, and it does not matter the order of the query.

**Named Windows**

An analytic function's OVER clause can reference a named window, which encapsulates one or more window clauses: a window partition (PARTITION BY) clause and (optionally) a window order (ORDER BY) clause. Named windows can be useful when you write queries that invoke
multiple analytic functions with similar OVER clause syntax—for example, they use the same partition clauses.

A query names a window as follows:

```sql
WINDOW window-name AS ( window-partition-clause [window-order-clause] );
```

The same query can name and reference multiple windows. All window names must be unique within the same query.

## Examples

The following query invokes two analytic functions, **RANK** and **DENSE_RANK**. Because the two functions use the same partition and order clauses, the query names a window `w` that specifies both clauses. The two functions reference this window as follows:

```sql
=> SELECT employee_region region, employee_key, annual_salary, 
    RANK() OVER w Rank,
    DENSE_RANK() OVER w "Dense Rank"
FROM employee_dimension WINDOW w AS (PARTITION BY employee_region ORDER BY annual_salary);
```

<table>
<thead>
<tr>
<th>region</th>
<th>employee_key</th>
<th>annual_salary</th>
<th>Rank</th>
<th>Dense Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>West</td>
<td>5248</td>
<td>1200</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>West</td>
<td>6880</td>
<td>1204</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>West</td>
<td>5700</td>
<td>1214</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>West</td>
<td>9857</td>
<td>1218</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>West</td>
<td>6014</td>
<td>1218</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>West</td>
<td>9221</td>
<td>1220</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>West</td>
<td>7646</td>
<td>1222</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>West</td>
<td>6621</td>
<td>1222</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>West</td>
<td>6488</td>
<td>1224</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>West</td>
<td>7659</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>7432</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>9905</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>9021</td>
<td>1228</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>West</td>
<td>7855</td>
<td>1228</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>West</td>
<td>7119</td>
<td>1230</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

If the named window omits an order clause, the query's OVER clauses can specify their own order clauses. For example, you can modify the previous query so each function uses a different order clause. The named window is defined so it includes only a partition clause:

```sql
=> SELECT employee_region region, employee_key, annual_salary, 
    RANK() OVER (w ORDER BY annual_salary DESC) Rank,
    DENSE_RANK() OVER (w ORDER BY annual_salary ASC) "Dense Rank"
FROM employee_dimension WINDOW w AS (PARTITION BY employee_region);
```

<table>
<thead>
<tr>
<th>region</th>
<th>employee_key</th>
<th>annual_salary</th>
<th>Rank</th>
<th>Dense Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>West</td>
<td>5248</td>
<td>1200</td>
<td>2795</td>
<td>1</td>
</tr>
<tr>
<td>West</td>
<td>6880</td>
<td>1204</td>
<td>2794</td>
<td>2</td>
</tr>
<tr>
<td>West</td>
<td>5700</td>
<td>1214</td>
<td>2793</td>
<td>3</td>
</tr>
<tr>
<td>West</td>
<td>6014</td>
<td>1218</td>
<td>2791</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Similarly, an OVER clause specifies a named window can also specify a window frame clause, provided the named window includes an order clause. This can be useful inasmuch as you cannot define a named windows to include a window frame clause.

For example, the following query defines a window that encapsulates partitioning and order clauses. The OVER clause invokes this window and also includes a window frame clause:

```sql
=> SELECT deptno, sal, empno, COUNT(*) OVER (w ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS count
    FROM emp WINDOW w AS (PARTITION BY deptno ORDER BY sal);
```

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

(11 rows)

Recursive Window References

A WINDOW clause can reference another window that is already named. For example, because named window w1 is defined before w2, the WINDOW clause that defines w2 can reference w1:

```sql
=> SELECT RANK() OVER(w1 ORDER BY sal DESC), RANK() OVER w2
    FROM emp WINDOW w1 AS (PARTITION BY deptno), w2 AS (w1 ORDER BY sal);
```
Restrictions

- An OVER clause can reference only one named window.
- Each WINDOW clause within the same query must have a unique name.
Analytic Query Examples

The following topics present use cases for analytic queries performing calculations:

Calculating a Median Value

A median is a numerical value that separates the higher half of a sample from the lower half. For example, you can retrieve the median of a finite list of numbers by arranging all observations from lowest value to highest value and then picking the middle one.

If the number of observations is even, then there is no single middle value; the median is the mean (average) of the two middle values.

The following example uses this table:

```sql
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT);
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

You can use the analytic function `MEDIAN` to calculate the median of all sales in this table. In the following query, the function's OVER clause is empty, so the query returns the same aggregated value for each row of the result set:

```sql
=> SELECT name, sales, MEDIAN(sales) OVER() AS median FROM allsales;
name | sales | median
-----+-------+-------
G   | 10    | 20    
C   | 15    | 20    
D   | 20    | 20    
B   | 20    | 20    
F   | 40    | 20    
E   | 50    | 20    
A   | 60    | 20    
(7 rows)
```

You can modify this query to group sales by state and obtain the median for each one. To do so, include a window partition clause in the OVER clause:

```sql
=> SELECT state, name, sales, MEDIAN(sales) OVER(partition by state) AS median FROM allsales;
state | name | sales | median
------+------+-------+-------
       |      |       |
(8 rows)
```
Getting Price Differential for Two Stocks

The following subquery selects out two stocks of interest. The outer query uses the LAST_VALUE() and OVER() components of analytics, with IGNORE NULLS.

Schema

```
DROP TABLE Ticks CASCADE;

CREATE TABLE Ticks (ts TIMESTAMP, Stock varchar(10), Bid float);
INSERT INTO Ticks VALUES('2011-07-12 10:23:54', 'abc', 10.12);
INSERT INTO Ticks VALUES('2011-07-12 10:23:58', 'abc', 10.34);
INSERT INTO Ticks VALUES('2011-07-12 10:23:59', 'abc', 10.75);
INSERT INTO Ticks VALUES('2011-07-12 10:25:15', 'abc', 11.98);
INSERT INTO Ticks VALUES('2011-07-12 10:25:16', 'abc');
INSERT INTO Ticks VALUES('2011-07-12 10:25:22', 'xyz', 45.16);
INSERT INTO Ticks VALUES('2011-07-12 10:25:27', 'xyz', 49.33);
INSERT INTO Ticks VALUES('2011-07-12 10:31:12', 'xyz', 65.25);
INSERT INTO Ticks VALUES('2011-07-12 10:31:15', 'xyz');

COMMIT;
```

ticks Table

```
== Select * from ticks;
   ts          | stock | bid
---------------+-------+-------
2011-07-12 10:23:59 | abc   | 10.75
2011-07-12 10:25:22 | xyz   | 45.16
2011-07-12 10:23:58 | abc   | 10.34
2011-07-12 10:25:27 | xyz   | 49.33
2011-07-12 10:31:15 | xyz   |
2011-07-12 10:25:15 | abc   | 11.98
2011-07-12 10:25:16 | abc   |
2011-07-12 10:31:12 | xyz   | 65.25
(9 rows)
```
Calculating the Moving Average

Calculating the moving average is useful to get an estimate about the trends in a data set. The moving average is the average of any subset of numbers over a period of time. For example, if you have retail data that spans over ten years, you could calculate a three year moving average, a four year moving average, and so on. This example calculates a 40-second moving average of bids for one stock. This example uses the `ticks` table schema.

Query

```sql
=> SELECT ts, bid, AVG(bid)
    OVER(ORDER BY ts 
    RANGE BETWEEN INTERVAL '40 seconds'
    PRECEDING AND CURRENT ROW)
FROM ticks
WHERE stock = 'abc'
GROUP BY bid, ts
ORDER BY ts;
```

| ts      | bid  | ?column?
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2011-07-12 10:23:58</td>
<td>10.34</td>
<td></td>
</tr>
<tr>
<td>2011-07-12 10:23:59</td>
<td>10.75</td>
<td></td>
</tr>
<tr>
<td>2011-07-12 10:25:15</td>
<td>11.98</td>
<td></td>
</tr>
<tr>
<td>2011-07-12 10:25:16</td>
<td>11.98</td>
<td></td>
</tr>
<tr>
<td>2011-07-12 10:25:22</td>
<td>45.16</td>
<td>-33.18</td>
</tr>
<tr>
<td>2011-07-12 10:25:27</td>
<td>49.33</td>
<td>-37.35</td>
</tr>
<tr>
<td>2011-07-12 10:31:12</td>
<td>65.25</td>
<td>-53.27</td>
</tr>
<tr>
<td>2011-07-12 10:31:15</td>
<td>65.25</td>
<td>-53.27</td>
</tr>
</tbody>
</table>
```

(9 rows)
Getting Latest Bid and Ask Results

The following query fills in missing NULL values to create a full book order showing latest bid and ask price and size, by vendor id. Original rows have values for (typically) one price and one size, so use `last_value_with "ignore nulls"` to find the most recent non-null value for the other pair each time there is an entry for the ID. `Sequenceno` provides a unique total ordering.

Schema:

```sql
CREATE TABLE bookorders(
  vendorid VARCHAR(100),
  date TIMESTAMP,
  sequenceno INT,
  askprice FLOAT,
  asksize INT,
  bidprice FLOAT,
  bidsize INT);
```
Query:

```sql
=> SELECT 
  sequenceno Seq, 
  date "Time", 
  vendorid ID, 
  LAST_VALUE (bidprice IGNORE NULLS) 
  OVER (PARTITION BY vendorid ORDER BY sequenceno 
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) 
  AS "Bid Price", 
  LAST_VALUE (bidsize IGNORE NULLS) 
  OVER (PARTITION BY vendorid ORDER BY sequenceno 
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) 
  AS "Bid Size", 
  LAST_VALUE (askprice IGNORE NULLS) 
  OVER (PARTITION BY vendorid ORDER BY sequenceno 
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) 
  AS "Ask Price", 
  LAST_VALUE (asksize IGNORE NULLS) 
  OVER (PARTITION BY vendorid ORDER BY sequenceno 
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) 
  AS "Ask Size" 
FROM bookorders 
ORDER BY sequenceno;
```

```
<table>
<thead>
<tr>
<th>Seq</th>
<th>Time</th>
<th>ID</th>
<th>Bid Price</th>
<th>Bid Size</th>
<th>Ask Price</th>
<th>Ask Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2011-07-12 10:23:54</td>
<td>3325XPK</td>
<td>10.23</td>
<td>59</td>
<td>10.12</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>2011-07-12 10:23:55</td>
<td>3345XPZ</td>
<td>10.75</td>
<td>57</td>
<td>10.55</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>2011-07-12 10:23:58</td>
<td>445XPKF</td>
<td>54</td>
<td>64</td>
<td>10.22</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>2011-07-12 10:23:59</td>
<td>3727XVK</td>
<td>11.9</td>
<td>66</td>
<td>11.87</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>2011-07-12 10:24:01</td>
<td>3325XYZ</td>
<td>15.1</td>
<td>59</td>
<td>15.05</td>
<td>44</td>
</tr>
<tr>
<td>7</td>
<td>2011-07-12 10:24:05</td>
<td>3675XVS</td>
<td>58</td>
<td>58</td>
<td>15.43</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>2011-07-12 10:25:15</td>
<td>8972VUG</td>
<td>15.11</td>
<td>57</td>
<td>14.95</td>
<td>52</td>
</tr>
</tbody>
</table>
```

(9 rows)

**Event-Based Windows**

Event-based windows let you break time series data into windows that border on significant events within the data. This is especially relevant in financial data where analysis often focuses on specific events as triggers to other activity.

Vertica provides two event-based window functions that are not part of the SQL-99 standard:

- **CONDITIONAL_CHANGE_EVENT** assigns an event window number to each row, starting from 0, and increments by 1 when the result of evaluating the argument expression on the current row differs from that on the previous row. This function is similar to the analytic
function **ROW_NUMBER**, which assigns a unique number, sequentially, starting from 1, to each row within a partition.

- **CONDITIONAL_TRUE_EVENT** assigns an event window number to each row, starting from 0, and increments the number by 1 when the result of the boolean argument expression evaluates true.

Both functions are described in greater detail below.

**Note:** **CONDITIONAL_CHANGE_EVENT** and **CONDITIONAL_TRUE_EVENT** do not support window framing.

**Example schema**

The examples on this page use the following schema:

```sql
CREATE TABLE TickStore3 (ts TIMESTAMP, symbol VARCHAR(8), bid FLOAT);
CREATE PROJECTION TickStore3_p (ts, symbol, bid) AS SELECT * FROM TickStore3 ORDER BY ts, symbol, bid UNSEGMENTED ALL NODES;
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:00', 'XYZ', 10.0);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:03', 'XYZ', 11.0);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:06', 'XYZ', 10.5);
INSERT INTO TickStore3 VALUES ('2009-01-01 03:00:09', 'XYZ', 11.0);
COMMIT;
```

**Using **CONDITIONAL_CHANGE_EVENT**

The analytical function **CONDITIONAL_CHANGE_EVENT** returns a sequence of integers indicating event window numbers, starting from 0. The function increments the event window number when the result of evaluating the function expression on the current row differs from the previous value.

In the following example, the first query returns all records from the TickStore3 table. The second query uses the **CONDITIONAL_CHANGE_EVENT** function on the bid column. Since each bid row value is different from the previous value, the function increments the window ID from 0 to 3:

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>11</td>
</tr>
<tr>
<td>2009-01-01 03:00:06</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
<tr>
<td>2009-01-01 03:00:09</td>
<td>XYZ</td>
<td>11</td>
</tr>
<tr>
<td>(4 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>cce</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2009-01-01 03:00:06</td>
<td>XYZ</td>
<td>10.5</td>
<td>2</td>
</tr>
<tr>
<td>2009-01-01 03:00:09</td>
<td>XYZ</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>(4 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following figure is a graphical illustration of the change in the bid price. Each value is different from its previous one, so the window ID increments for each time slice:

So the window ID starts at 0 and increments at every change in from the previous value.

In this example, the bid price changes from $10 to $11 in the second row, but then stays the same. CONDITIONAL_CHANGE_EVENT increments the event window ID in row 2, but not subsequently:

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>cce</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2009-01-01 03:00:06</td>
<td>XYZ</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2009-01-01 03:00:09</td>
<td>XYZ</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

The following figure is a graphical illustration of the change in the bid price at 3:00:03 only. The price stays the same at 3:00:06 and 3:00:09, so the window ID remains at 1 for each time slice after the change:
Using CONDITIONAL_TRUE_EVENT

Like CONDITIONAL_CHANGE_EVENT, the analytic function CONDITIONAL_TRUE_EVENT also returns a sequence of integers indicating event window numbers, starting from 0. The two functions differ as follows:

- CONDITIONAL_TRUE_EVENT increments the window ID each time its expression evaluates to true.
- CONDITIONAL_CHANGE_EVENT increments on a comparison expression with the previous value.

In the following example, the first query returns all records from the TickStore3 table. The second query uses CONDITIONAL_TRUE_EVENT to test whether the current bid is greater than a given value (10.6). Each time the expression tests true, the function increments the window ID. The first time the function increments the window ID is on row 2, when the value is 11. The expression tests false for the next row (value is not greater than 10.6), so the function does not increment the event window ID. In the final row, the expression is true for the given condition, and the function increments the window:

```sql
SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;
SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6) OVER(ORDER BY ts) FROM Tickstore3;
```

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>cte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>3:00:03</td>
<td></td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>3:00:06</td>
<td></td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td>3:00:09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:00:12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following figure is a graphical illustration that shows the bid values and window ID changes. Because the bid value is greater than $10.6 on only the second and fourth time slices (3:00:03 and 3:00:09), the window ID returns <0,1,2>:

In the following example, the first query returns all records from the TickStore3 table, ordered by the tickstore values (ts). The second query uses CONDITIONAL_TRUE_EVENT to increment the window ID each time the bid value is greater than 10.6. The first time the function increments the event window ID is on row 2, where the value is 11. The window ID then increments each time after that, because the expression (bid > 10.6) tests true for each time slice:

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>11</td>
</tr>
<tr>
<td>2009-01-01 03:00:06</td>
<td>XYZ</td>
<td>11</td>
</tr>
<tr>
<td>2009-01-01 03:00:09</td>
<td>XYZ</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>cte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2009-01-01 03:00:06</td>
<td>XYZ</td>
<td>10.5</td>
<td>1</td>
</tr>
<tr>
<td>2009-01-01 03:00:09</td>
<td>XYZ</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;

SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6) OVER(OVER BY ts) FROM Tickstore3;
The following figure is a graphical illustration that shows the bid values and window ID changes. The bid value is greater than 10.6 on the second time slice (3:00:03) and remains for the remaining two time slices. The function increments the event window ID each time because the expression tests true:

![Graphical Illustration]

Advanced Use of Event-Based Windows

In event-based window functions, the condition expression accesses values from the current row only. To access a previous value, you can use a more powerful event-based window that allows the window event condition to include previous data points. For example, analytic function \( \text{LAG}(x, n) \) retrieves the value of column \( x \) in the \( n \)th last input record. In this case, \( \text{LAG} \) shares the \text{OVER} specifications of the \text{CONDITIONAL\_CHANGE\_EVENT} or \text{CONDITIONAL\_TRUE\_EVENT} function expression.

In the following example, the first query returns all records from the TickStore3 table. The second query uses \text{CONDITIONAL\_TRUE\_EVENT} with the \text{LAG} function in its boolean expression. In this case, \text{CONDITIONAL\_TRUE\_EVENT} increments the event window ID each time the bid value on the current row is less than the previous value. The first time \text{CONDITIONAL\_TRUE\_EVENT} increments the window ID starts on the third time slice, when the expression tests true. The current value (10.5) is less than the previous value. The window ID is not incremented in the last row because the final value is greater than the previous row:

<table>
<thead>
<tr>
<th>Query 1</th>
<th>Query 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ts, symbol, bid FROM Tickstore3 ORDER BY ts;</td>
<td>SELECT CONDITIONAL_TRUE_EVENT(bid &lt; LAG(bid)) OVER(ORDER BY ts) FROM Tickstore;</td>
</tr>
</tbody>
</table>
The following figure illustrates the second query above. When the bid price is less than the previous value, the window ID gets incremented, which occurs only in the third time slice (3:00:06):

See Also

- Sessionization with Event-Based Windows
- Time Series Analytics

Sessionization with Event-Based Windows

Sessionization, a special case of event-based windows, is a feature often used to analyze click streams, such as identifying web browsing sessions from recorded web clicks.

In Vertica, given an input clickstream table, where each row records a Web page click made by a particular user (or IP address), the sessionization computation attempts to identify Web
browsing sessions from the recorded clicks by grouping the clicks from each user based on the time-intervals between the clicks. If two clicks from the same user are made too far apart in time, as defined by a time-out threshold, the clicks are treated as though they are from two different browsing sessions.

Example Schema

The examples in this topic use the following WebClicks schema to represent a simple clickstream table:

```sql
CREATE TABLE WebClicks(userId INT, timestamp TIMESTAMP);
INSERT INTO WebClicks VALUES (1, '2009-12-08 15:00:00 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 15:00:25 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 15:00:45 pm');
INSERT INTO WebClicks VALUES (1, '2009-12-08 15:01:45 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 15:02:45 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 15:02:55 pm');
INSERT INTO WebClicks VALUES (2, '2009-12-08 15:03:55 pm');
COMMIT;
```

The input table WebClicks contains the following rows:

```
<table>
<thead>
<tr>
<th>userId</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2009-12-08 15:00:00</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:25</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:45</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:01:45</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:02:45</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:02:55</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:03:55</td>
</tr>
</tbody>
</table>
```

In the following query, sessionization performs computation on the SELECT list columns, showing the difference between the current and previous timestamp value using LAG(). It evaluates to true and increments the window ID when the difference is greater than 30 seconds.

```sql
=> SELECT userId, timestamp, 
    CONDITIONAL_TRUE_EVENT(timestamp - LAG(timestamp) > '30 seconds') 
OVER(PARTITION BY userId ORDER BY timestamp) AS session FROM WebClicks;
```

```
<table>
<thead>
<tr>
<th>userId</th>
<th>timestamp</th>
<th>session</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2009-12-08 15:00:00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:25</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:45</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:01:45</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:02:45</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:02:55</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:03:55</td>
<td>1</td>
</tr>
</tbody>
</table>
```
In the output, the session column contains the window ID from the CONDITIONAL_TRUE_EVENT function. The window ID evaluates to true on row 4 (timestamp 15:01:45), and the ID that follows row 4 is zero because it is the start of a new partition (for user ID 2), and that row does not evaluate to true until the last line in the output.

You might want to give users different time-out thresholds. For example, one user might have a slower network connection or be multi-tasking, while another user might have a faster connection and be focused on a single Web site, doing a single task.

To compute an adaptive time-out threshold based on the last 2 clicks, use CONDITIONAL_TRUE_EVENT with LAG to return the average time between the last 2 clicks with a grace period of 3 seconds:

```
=> SELECT userId, timestamp, CONDITIONAL_TRUE_EVENT(timestamp - LAG(timestamp) 
(LAG(timestamp, 1) - LAG(timestamp, 3)) / 2 + '3 seconds') 
OVER(PARTITION BY userId ORDER BY timestamp) AS session 
FROM WebClicks;
```

<table>
<thead>
<tr>
<th>userId</th>
<th>timestamp</th>
<th>session</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2009-12-08 15:02:45</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:02:55</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2009-12-08 15:03:55</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:25</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:00:45</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2009-12-08 15:01:45</td>
<td>1</td>
</tr>
</tbody>
</table>

(7 rows)

Note: You cannot define a moving window in time series data. For example, if the query is evaluating the first row and there’s no data, it will be the current row. If you have a lag of 2, no results are returned until the third row.

See Also

- Event-Based Windows
- CONDITIONAL_TRUE_EVENT [Analytic]
Machine Learning for Predictive Analytics

Vertica provides a number of machine learning functions for performing in-database analysis. These functions can perform a number of data preparation and predictive tasks—for example:

- Balance an uneven distribution of classes: see Balancing Imbalanced Data for an example.
- Remove outliers from your data: see Detecting Outliers for an example.
- Impute missing values in a data set: see Imputing Missing Values for an example.
- Normalize data to organize different scales of numeric data to an equivalent scale: see Normalizing Data for an example.
- Create a sample of a larger data set: see Sampling Data for an example.
- Use regression algorithms to make predictions about features in your data set and an observed value response: see for more information.
- Use classification algorithms to assign items in a data set to different categories: see Classification Algorithms for more information.
- Use the clustering algorithm to partition data using k-means clustering: see Clustering Algorithms for more information.

For more information about specific machine learning functions see Machine Learning Functions.

Downloading the Machine Learning Example Data

You need several data sets to run the machine learning examples. You can download these data sets from the Vertica Github repository. These examples introduce the machine learning functionality provide by Vertica.

You can download the example data in either of two ways:
• Download the ZIP file. Extract the contents of the file into a directory.

• Clone the Vertica Machine Learning Github repository. Using a terminal window, run the following command:

   ```
   $ git clone https://github.com/vertica/Machine-Learning-Examples
   ```

### Loading the Example Data

You can load the example data by either:

• Copying and pasting the DDL and DML operations in `load_ml_data.sql` in a vsql prompt or another Vertica client.

• Running the following command from a terminal window in the Machine-Learning-Examples directory:

   ```
   $ /opt/vertica/bin/vsql -d <name of your database> -f load_ml_data.sql
   ```

### Example Data Descriptions

The repository contains the following data sets.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agar_dish</td>
<td>Synthetic data set meant to represent clustering of bacteria on an agar dish. Contains the following columns: id, x-coordinate, and y-coordinate.</td>
</tr>
<tr>
<td>agar_dish_2</td>
<td>125 rows sampled randomly from the original 500 rows of the agar_dish data set.</td>
</tr>
<tr>
<td>agar_dish_1</td>
<td>375 rows sampled randomly from the original 500 rows of the agar_dish data set.</td>
</tr>
<tr>
<td>baseball</td>
<td>Contains statistics from a fictional baseball league. The statistics included are: first name, last name, date of birth, team name, homeruns, hits, batting average, and salary.</td>
</tr>
<tr>
<td>faithful</td>
<td>Wait times between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>faithful_testing</td>
<td>Roughly 60% of the original 272 rows of the faithful data set.</td>
</tr>
<tr>
<td>faithful_training</td>
<td>Roughly 40% of the original 272 rows of the faithful data set.</td>
</tr>
<tr>
<td>iris1</td>
<td>90 rows sampled randomly from the original 150 rows in the iris data set.</td>
</tr>
<tr>
<td>iris2</td>
<td>60 rows sampled randomly from the original 150 rows in the iris data set.</td>
</tr>
</tbody>
</table>
### Data Preparation

Before you can analyze your data, you must prepare it. You can do the following data preparation tasks in Vertica:

- **Balancing Imbalanced Data**
- **Detecting Outliers**
- **Encoding Categorical Columns**
- **Imputing Missing Values**
- **Normalizing Data**
- **Sampling Data**

### Balancing Imbalanced Data

Imbalanced data occurs when an uneven distribution of classes occurs in the data. You see imbalanced data a lot in financial transaction data where the majority of the transactions are not fraudulent and a small number of the transactions are fraudulent. Building a predictive model on the imbalanced data set would cause a model that appears to yield high accuracy but does not generalize well to the new data in the minority class. To prevent creating models with false levels of accuracy, you should rebalance your imbalanced data before creating a predictive model.

Before you begin the example, make sure that you have [loaded the Machine Learning sample data](#).

The following example shows you how to use the BALANCE function to create a more balanced data set.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>salary_data</td>
<td>Contains fictional employee data. The data included are: employee id, first name, last name, years worked, and current salary.</td>
</tr>
<tr>
<td>transaction_data</td>
<td>Contains fictional credit card transactions with a BOOLEAN column indicating whether there was fraud associated with the transaction. The data included are: first name, last name, store, cost, and fraud.</td>
</tr>
</tbody>
</table>
1. **View the distribution of the classes.**

   ```sql
   SELECT fraud, COUNT(fraud) FROM transaction_data GROUP BY fraud;
   fraud | COUNT
   ------|------
   TRUE  | 19
   FALSE | 981
   (2 rows)
   ```

2. **Use the BALANCE function to create a more balanced data set.**

   ```sql
   SELECT BALANCE('balance_fin_data', 'transaction_data', 'fraud', 'under_sampling'
                  USING PARAMETERS sampling_ratio = 0.2);
   BALANCE
   ------------
   Finished in 1 iteration
   (1 row)
   ```

3. **View the new distribution of the classifiers.**

   ```sql
   SELECT fraud, COUNT(fraud) FROM balance_fin_data GROUP BY fraud;
   fraud | COUNT
   ------|------
   t     | 19
   f     | 236
   (2 rows)
   ```

**See Also**

- **BALANCE**

**Detecting Outliers**

Before you perform an in-depth analysis of your data, you should first remove the outliers from the data. Outliers are data points that greatly differ from other similar data points. If you leave outliers in your data, then you risk misclassifying data, introducing bias, or incorrect calculations.

For this example we will use the baseball data set found on the Vertica GitHub page.

1. **Create a table.**

   ```sql
   CREATE TABLE baseball (id identity, first_name varchar(50), last_name varchar(50), dob DATE,
                           team varchar(20), hr int, hits int, avg float, salary float);
   ```
CREATE TABLE

2. Load the data.

=> COPY baseball FROM 'baseball.csv' DELIMITER ',';
   Rows Loaded
   ---------------
   1000
   (1 row)

3. Detect the outliers based on the hr, hits, and salary columns. We will use the id and team columns as our key columns. The DETECT_OUTLIERS function will create a view containing the outliers with the input and key columns. Before you use the DETECT_OUTLIERS function, make sure that you are a superuser or have CREATE privileges for the schema and SELECT privileges for the table.

=> SELECT DETECT_OUTLIERS('baseball_hr_hits_salary_outliers', 'baseball', 'hr, hits, salary', 'robust_zscore')
   USING PARAMETERS outlier_threshold=3.0);

DETECT_OUTLIERS
---------------------------
Detected 5 outliers
(1 row)

4. Query the output view containing the outliers.

=> SELECT * FROM baseball_hr_hits_salary_outliers;

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>dob</th>
<th>team</th>
<th>hr</th>
<th>hits</th>
<th>avg</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>Marie</td>
<td>Fields</td>
<td>1985-11-23</td>
<td>Mauv</td>
<td>888</td>
<td>34</td>
<td>0.283</td>
<td>999999999341471e+16</td>
</tr>
<tr>
<td>89</td>
<td>Jacqueline</td>
<td>Richards</td>
<td>1975-10-06</td>
<td>Pink</td>
<td>273333</td>
<td>4490260</td>
<td>0.324</td>
<td>4444444444828e+17</td>
</tr>
<tr>
<td>87</td>
<td>Jose</td>
<td>Stephens</td>
<td>1991-07-20</td>
<td>Green</td>
<td>80</td>
<td>64253</td>
<td>0.69</td>
<td>16832567.12</td>
</tr>
<tr>
<td>222</td>
<td>Gerald</td>
<td>Fuller</td>
<td>1991-02-13</td>
<td>Goldenrod</td>
<td>3200000</td>
<td>216</td>
<td>0.299</td>
<td>37008899.76</td>
</tr>
<tr>
<td>147</td>
<td>Debra</td>
<td>Hall</td>
<td>1980-12-31</td>
<td>Maroon</td>
<td>1100037</td>
<td>230</td>
<td>0.431</td>
<td>9000101403</td>
</tr>
</tbody>
</table>

(5 rows)

5. Create a view omitting the outliers from the table.

=> CREATE VIEW clean_baseball AS
   SELECT * FROM baseball WHERE id NOT IN (SELECT id FROM baseball_hr_hits_salary_outliers);
CREATE VIEW

6. Perform your analysis.
See Also

- DETECT_OUTLIERS
- LINEAR_REG
- LOGISTIC_REG

Encoding Categorical Columns

Many machine learning algorithms cannot work with categorical data. To accommodate such algorithms, categorical data must be converted to numerical data before training. Directly mapping the categorical values into indices is not enough. For example, if your categorical feature has three distinct values "red", "green" and "blue", replacing them with 1, 2 and 3 may have a negative impact on the training process because algorithms usually rely on some kind of numerical distances between values to discriminate between them. In this case, the Euclidean distance from 1 to 3 is twice the distance from 1 to 2, which means the training process will think that "red" is much more different than "blue", while it is more similar to "green". Alternatively, one hot encoding maps each categorical value to a binary vector to avoid this problem. For example, "red" can be mapped to [1,0,0], "green" to [0,1,0] and "blue" to [0,0,1]. Now, the pair-wise distances between the three categories are all the same. One hot encoding allows you to convert categorical variables to binary values so that you can use different machine learning algorithms to evaluate your data.

The following example shows how you can apply one hot encoding to the Titanic data set. If you would like to read more about this data set, see the Kaggle site.

Suppose you want to use a logistic regression classifier to predict which passengers survived the sinking of the Titanic. You cannot use categorical features for logistic regression without one hot encoding. This data set has two categorical features that you can use. The "sex" feature can be either male or female. The "embarkation_point" feature can be one of the following:

- S for Southampton
- Q for Queenstown
- C for Cherbourg
Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Run the `ONE_HOT_ENCODERFIT` function on the training data:

```
=> SELECT ONE_HOT_ENCODERFIT('titanic_encoder', 'titanic_training', 'sex, embarkation_point);
ONE_HOT_ENCODERFIT
------------------
Success
(1 row)
```

2. View a summary of the `titanic_encoder` model:

```
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='titanic_encoder');
GET_MODEL_SUMMARY
----------------------------------------------------------------------------------
-----------
call_string
-----------
SELECT one_hot_encoder_fit('public.titanic_encoder','titanic_training','sex, embarkation_point'
USING PARAMETERS exclude_columns='', output_view='', extra_levels='{}');
-----------
categories
-----------
category_name | category_level| category_level_index
-------------------|------------------|------------------
embarkation_point| C | 0
embarkation_point| Q | 1
embarkation_point| S | 2
embarkation_point| | 3
sex | female | 0
sex | male | 1
(1 row)
```

3. Run the `GET_MODEL_ATTRIBUTE` function. This function returns the categorical levels in their native data types, so they can be compared easily with the original table:

```
=> SELECT * FROM (SELECT GET_MODEL_ATTRIBUTE(USING PARAMETERS model_name='titanic_encoder',
attr_name='varchar_categories')) AS attrs INNER JOIN (SELECT passenger_id, name, sex, age,
embarkation_point FROM titanic_training) AS original_data ON attrs.category_level
LIKE original_data.embarkation_point ORDER BY original_data.passenger_id LIMIT 10;
category_name | category_level | category_level_index | passenger_id | name
| sex | age | embarkation_point
-------------------|------------------|------------------|------------------|------------------
embarkation_point | S | 2 | 1 | Braund, Mr. Owen Harris
| male | 22 | S
embarkation_point | C | 0 | 2 | Cumings, Mrs. John Bradley
(Florence Briggs Thayer | female | 38 | C
embarkation_point | S | 2 | 3 | Heikkinen, Miss. Laina
| female | 26 | S
embarkation_point | S | 2 | 4 | Futrelle, Mrs. Jacques
Heath
```
4. Run the APPLY_ONE_HOT_ENCODER function on both the training and testing data:

```sql
=> CREATE VIEW titanic_training_encoded AS SELECT passenger_id, survived, pclass, sex_1, age, sibling_and_spouse_count, parent_and_child_count, fare, embarkation_point_1, embarkation_point_2 FROM (SELECT APPLY_ONE_HOT_ENCODER(* USING PARAMETERS model_name='titanic_encoder') FROM titanic_training) AS sq;
```

```sql
CREATE VIEW
```

```sql
=> CREATE VIEW titanic_testing_encoded AS SELECT passenger_id, name, pclass, sex_1, age, sibling_and_spouse_count, parent_and_child_count, fare, embarkation_point_1, embarkation_point_2 FROM (SELECT APPLY_ONE_HOT_ENCODER(* USING PARAMETERS model_name='titanic_encoder') FROM titanic_testing) AS sq;
```

5. Then, train a logistic regression classifier on the training data, and execute the model on the testing data:

```sql
=> SELECT LOGISTIC_REG('titanic_log_reg', 'titanic_training_encoded', 'survived', '**' USING PARAMETERS exclude_columns='passenger_id, survived');

LOGISTIC_REG

Finished in 5 iterations
(1 row)
```

```sql
=> SELECT passenger_id, name, PREDICT_LOGISTIC_REG(pclass, sex_1, age, sibling_and_spouse_count, parent_and_child_count, fare, embarkation_point_1, embarkation_point_2 USING PARAMETERS model_name='titanic_log_reg') FROM titanic_testing_encoded ORDER BY passenger_id LIMIT 10;
```

<table>
<thead>
<tr>
<th>passenger_id</th>
<th>name</th>
<th>PREDICT_LOGISTIC_REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>893</td>
<td>Wilkes, Mrs. James (Ellen Needs)</td>
<td>0</td>
</tr>
<tr>
<td>894</td>
<td>Myles, Mr. Thomas Francis</td>
<td>0</td>
</tr>
<tr>
<td>895</td>
<td>Wirz, Mr. Albert</td>
<td>0</td>
</tr>
<tr>
<td>896</td>
<td>Hirvonen, Mrs. Alexander (Helga E Lindqvist)</td>
<td>1</td>
</tr>
<tr>
<td>897</td>
<td>Svensson, Mr. Johan Cervin</td>
<td>0</td>
</tr>
<tr>
<td>898</td>
<td>Connolly, Miss. Kate</td>
<td>1</td>
</tr>
<tr>
<td>899</td>
<td>Caldwell, Mr. Albert Francis</td>
<td>0</td>
</tr>
<tr>
<td>900</td>
<td>Abraham, Mrs. Joseph (Sophie Halaut Easu)</td>
<td>1</td>
</tr>
<tr>
<td>901</td>
<td>Davies, Mr. John Samuel</td>
<td>0</td>
</tr>
<tr>
<td>902</td>
<td>Ilieff, Mr. Ylio</td>
<td></td>
</tr>
</tbody>
</table>
Imputing Missing Values

You can use the IMPUTE function to impute missing values in your data set. The function helps you impute missing values with plausible data values. This impute example uses the small_input_impute table. Using the function, you can specify either the mean or mode method.

These examples show how you can use the IMPUTE function on the small_input_impute table.

Create the Table

First, create the small_input_impute table:

```sql
=> CREATE TABLE small_input_impute(pid int, pclass int, gender int, x1 float, x2 float, x3 float, x4 INT,
   x5 char, x6 varchar);
```

Then insert the table values:

```sql
=> INSERT INTO small_input_impute VALUES( 1, 0, 0, -9.445818, -9.740541, -9.786974, 3, 't', 'A');
=> INSERT INTO small_input_impute VALUES( 2, 0, 0, -9.618292, -9.308881, -9.562255, 4, 't', 'A');
=> INSERT INTO small_input_impute VALUES( 3, 0, 0, -9.060605, -9.390844, -9.559848, 6, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 4, 0, 0, -2.264599, -2.615146, -2.107290, 15, 't', 'B');
=> INSERT INTO small_input_impute VALUES( 5, 0, 1, -2.590837, -2.892819, -2.702960, 2, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 6, 0, 1, -2.264599, -2.615146, -2.107290, 11, 't', 'C');
=> INSERT INTO small_input_impute VALUES( 7, 1, 1, 3.829239, 3.007650, 'INFINITY', NULL, 'f', 'C');
=> INSERT INTO small_input_impute VALUES( 8, 1, 1, 3.273592, NULL, 3.477332, 18, 'f', 'B');
=> INSERT INTO small_input_impute VALUES( 9, 1, 1, NULL, 3.841606, 3.754375, 20, 'f', 'B');
=> INSERT INTO small_input_impute VALUES(10, 1, 1, NULL, 3.841606, 3.754375, 20, 't', 'A');
=> INSERT INTO small_input_impute VALUES(11, 0, 0, -9.445818, -9.740541, -9.786974, 3, 't', 'B');
=> INSERT INTO small_input_impute VALUES(12, 0, 0, -9.618292, -9.308881, -9.562255, 4, 't', 'C');
=> INSERT INTO small_input_impute VALUES(13, 0, 0, -9.060605, -9.390844, -9.559848, 6, 't', 'C');
=> INSERT INTO small_input_impute VALUES(14, 0, 0, -2.264599, -2.615146, -2.107290, 15, 'f', 'A');
=> INSERT INTO small_input_impute VALUES(15, 0, 1, -2.590837, -2.892819, -2.702960, 2, 'f', 'A');
=> INSERT INTO small_input_impute VALUES(16, 0, 1, -2.264599, -2.615146, -2.107290, 11, 'f', 'A');
=> INSERT INTO small_input_impute VALUES(17, 1, 1, 3.829239, 3.007650, 'INFINITY', NULL, 'f', 'B');
=> INSERT INTO small_input_impute VALUES(18, 1, 1, 3.273592, NULL, 3.477332, 18, 't', 'B');
=> INSERT INTO small_input_impute VALUES(19, 1, 1, NULL, 3.841606, 3.754375, 20, 't', NULL);
=> INSERT INTO small_input_impute VALUES(20, 1, 1, NULL, 3.841606, 3.754375, 20, NULL, 'C');
```

You can see the table with the missing values:
Specify the Mean Method

Execute the IMPUTE function, specifying the mean method, without using the partition_columns parameter:

```sql
=> SELECT IMPUTE('output_view','small_input_impute', 'pid, x1,x2,x3,x4','mean' USING PARAMETERS exclude_columns='pid');

IMPUTE

----------------------------------------
Finished in 1 iteration
(1 row)
```

View output_view to see the imputed values:

```sql
=> SELECT * FROM output_view;
pid | pclass | gender | x1   | x2   | x3   | x4   | x5   | x6
----------------------------------------
5   | 0      | 1      | -2.590837 | -2.892819 | -2.70296 | 2 | t | C
7   | 1      | 1      | 3.829239 | 3.08765 | -3.12989705263158 | 11 | f | C
13  | 0      | 0      | -9.060605 | -9.390844 | -9.559848 | 6 | t | C
15  | 0      | 1      | -2.590837 | -2.892819 | -2.70296 | 2 | f | A
16  | 0      | 1      | -2.264599 | -2.615146 | -2.10729 | 11 | f | A
19  | 1      | 1      | 3.841606 | 3.08765 | Infinity | f | B
1    | 0      | 0     | -9.445818 | -9.740541 | -3.841606 | 3.477332 | 18 | f | B
18  | 1      | 1      | 3.841606 | 3.08765 | Infinity | f | B
20  | 1      | 1      | 3.841606 | 3.08765 | Infinity | f | B
9    | 1      | 1     | 3.841606 | 3.08765 | Infinity | f | B
11  | 0      | 0     | -9.445818 | -9.740541 | -9.786974 | 3 | t | B
12  | 0      | 0     | -9.618292 | -9.308881 | -9.562255 | 4 | t | B
14  | 0      | 0     | -2.624599 | -2.615146 | -2.10729 | 15 | f | A
17  | 1      | 1      | 3.829239 | 3.08765 | Infinity | f | B
```

(21 rows)
You can also execute the IMPUTE function, specifying the mean method and using the partition_columns parameter. This parameter uses the GROUP BY clause:

```sql
=> SELECT IMPUTE('output_view_group','small_input_impute', 'pid, x1,x2,x3,x4','mean' USING PARAMETERS exclude_columns='pid', partition_columns='pclass,gender');

Finished in 1 iteration

(1 row)
```

View `output_view_group` to see the imputed values:

```sql
=> SELECT * FROM output_view_group;

<table>
<thead>
<tr>
<th>pid</th>
<th>pclass</th>
<th>gender</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>-2.590837</td>
<td>-2.892819</td>
<td>-2.70296</td>
<td>2</td>
<td>t</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3.829239</td>
<td>3.08765</td>
<td>-3.12980702563158</td>
<td>11</td>
<td>f</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-9.445818</td>
<td>-9.740541</td>
<td>-9.786974</td>
<td>3</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-9.445818</td>
<td>-9.740541</td>
<td>-9.786974</td>
<td>3</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-9.618292</td>
<td>-9.308881</td>
<td>-9.562255</td>
<td>4</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-9.606085</td>
<td>-9.390844</td>
<td>-9.559848</td>
<td>6</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>15</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>15</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>3.273592</td>
<td>3.22766163157895</td>
<td>3.477332</td>
<td>18</td>
<td>f</td>
<td>B</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>-3.86645035294118</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>1</td>
<td>3.273592</td>
<td>-3.22766163157895</td>
<td>3.477332</td>
<td>18</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>-3.86645035294118</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>t</td>
<td>C</td>
</tr>
</tbody>
</table>

(21 rows)
```
Specify the Mode Method

Execute the **impute** function, specifying the mode method, without using the **partition_columns** parameter:

```sql
=> SELECT impute('output_view_mode','small_input_impute', 'pid, x5,x6','mode'
    USING PARAMETERS exclude_columns='pid');
```

impute
-------------------
Finished in 1 iteration
(1 row)

View **output_view_mode** to see the imputed values:

```sql
=> SELECT * FROM output_view_mode;
pid | pclass | gender | x1 | x2 | x3 | x4 | x5 | x6  
-----+--------+--------+----+----+----+----+----+----
  5   |  0     |  1     | 0.2590837 | 2.892819 | 2.70296 | 2 | t | C
  7   |  1     |  1     | 3.829239  | 3.08765  | Infinity | f | C
 13   |  0     |  0     | -9.60605  | -9.390844 | -9.559848 | 6 | t | C
 15   |  0     |  1     | -2.590837 | -2.892819 | -2.70296 | 2 | f | A
 16   |  0     |  1     | -2.264599 | -2.615146 | -2.10729 | 11 | f | A
 19   |  1     |  1     |  3.841606 |  3.754375 |  3.754375 | 18 | t | B
  1   |  0     |  0     | -9.445818 | -9.740541 | -9.786974 | 3 | t | A
  1   |  0     |  0     | -9.445818 | -9.740541 | -9.786974 | 3 | t | A
  2   |  0     |  0     | -9.618292 | -9.308881 | -9.562255 | 4 | t | A
  3   |  0     |  0     | -9.60605  | -9.390844 | -9.559848 | 6 | t | B
  4   |  0     |  0     | -2.264599 | -2.615146 | -2.10729 | 15 | t | B
  6   |  0     |  1     | -2.264599 | -2.615146 | -2.10729 | 11 | t | C
  8   |  1     |  1     |  3.273592  |  3.477332 |  3.754375 | 18 | f | B
 10   |  1     |  1     |  3.841606 |  3.754375 |  3.754375 | 20 | t | A
 18   |  1     |  1     |  3.273592  |  3.477332 |  3.754375 | 18 | t | B
 20   |  1     |  1     |  3.841606 |  3.754375 |  3.754375 | 20 | t | C
  9   |  1     |  1     |  3.841606 |  3.754375 |  3.754375 | 20 | f | B
 11   |  0     |  0     | -9.445818 | -9.740541 | -9.786974 | 3 | t | B
 12   |  0     |  0     | -9.618292 | -9.308881 | -9.562255 | 4 | t | C
 14   |  0     |  0     | -2.264599 | -2.615146 | -2.10729 | 15 | f | A
 17   |  1     |  1     |  3.829239 |  3.08765  | Infinity | f | B
(21 rows)
```

You can also execute the **impute** function, specifying the mode method and using the **partition_columns** parameter. This parameter uses the **GROUP BY** clause:

```sql
=> SELECT impute('output_view_mode_group','small_input_impute','pid, x5,x6','mode'
    USING PARAMETERS exclude_columns='pid',partition_columns='pclass,gender');
```

impute
-------------------
Finished in 1 iteration
(1 row)

View **output_view_mode_group** to see the imputed values:


=> SELECT * FROM output_view_mode_group;

<table>
<thead>
<tr>
<th>pid</th>
<th>pclass</th>
<th>gender</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-9.445818</td>
<td>-9.740541</td>
<td>-9.786974</td>
<td>3</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-9.445818</td>
<td>-9.740541</td>
<td>-9.786974</td>
<td>3</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-9.618292</td>
<td>-9.308881</td>
<td>-9.562255</td>
<td>4</td>
<td>t</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-9.060805</td>
<td>-9.390844</td>
<td>-9.559848</td>
<td>6</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>15</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>-9.060805</td>
<td>-9.390844</td>
<td>-9.559848</td>
<td>6</td>
<td>t</td>
<td>C</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>-9.445818</td>
<td>-9.740541</td>
<td>-9.786974</td>
<td>3</td>
<td>t</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>-9.618292</td>
<td>-9.308881</td>
<td>-9.562255</td>
<td>4</td>
<td>t</td>
<td>C</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>15</td>
<td>f</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>-2.590837</td>
<td>-2.892819</td>
<td>-2.70296</td>
<td>2</td>
<td>t</td>
<td>C</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>-2.590837</td>
<td>-2.892819</td>
<td>-2.70296</td>
<td>2</td>
<td>f</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>11</td>
<td>f</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>-2.264599</td>
<td>-2.615146</td>
<td>-2.10729</td>
<td>11</td>
<td>t</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3.829239</td>
<td>3.08765</td>
<td>Infinity</td>
<td>f</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>1</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>t</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>f</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>3.829239</td>
<td>3.08765</td>
<td>Infinity</td>
<td>f</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>3.273592</td>
<td>3.477332</td>
<td>18</td>
<td>f</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>t</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>1</td>
<td>3.273592</td>
<td>3.477332</td>
<td>18</td>
<td>t</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>3.841606</td>
<td>3.754375</td>
<td>20</td>
<td>f</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

(21 rows)

See Also

IMPUTE

Normalizing Data

The purpose of normalization is, primarily, to scale numeric data from different columns down to an equivalent scale. For example, suppose you execute the LINEAR_REG function on a data set with two feature columns, "current_salary" and "years_worked." The output value you are trying to predict is a worker's future salary. The values in the "current_salary" column are likely to have a far wider range, and much larger values, than the values in the "years_worked" column. Therefore, the values in the "current_salary" column can overshadow the values in the "years_worked" column, thus skewing your model.

Vertica offers the following data preparation methods which use normalization. These methods are:

- MinMax
  Using the MinMax normalization method, you can normalize the values in both of these
columns to be within a distribution of values between 0 and 1. Doing so allows you to compare values on very different scales to one another by reducing the dominance of one column over the other.

- **Z-score**
  Using the Z-score normalization method, you can normalize the values in both of these columns to be the number of standard deviations an observation is from the mean of each column. This allows you to compare your data to a normally distributed random variable.

- **Robust Z-score**
  Using the Robust Z-score normalization method, you can lessen the influence of outliers on Z-score calculations. Robust Z-score normalization uses the median value as opposed to the mean value used in Z-score. By using the median instead of the mean, it helps remove some of the influence of outliers in the data.

Normalizing data results in the creation of a view where the normalized data is saved. The output_view option in the `NORMALIZE` function determines name of the view.

### Normalizing Salary Data Using MinMax

The following example shows how you can normalize the `salary_data` table using the MinMax normalization method.

Before you begin the example, make sure that you have loaded the `Machine Learning sample data`.

```sql
=> SELECT NORMALIZE('normalized_salary_data', 'salary_data', 'current_salary, years_worked', 'minmax');

---
NORMALIZE
---
Finished in 1 iteration
(1 row)

=> SELECT * FROM normalized_salary_data;

<table>
<thead>
<tr>
<th>employee_id</th>
<th>first_name</th>
<th>last_name</th>
<th>years_worked</th>
<th>current_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>189</td>
<td>Shawn</td>
<td>Moore</td>
<td>0.3500000000000000</td>
<td>0.437246565765357217</td>
</tr>
<tr>
<td>518</td>
<td>Earl</td>
<td>Shaw</td>
<td>0.1000000000000000</td>
<td>0.97867411144492943</td>
</tr>
<tr>
<td>1126</td>
<td>Susan</td>
<td>Alexander</td>
<td>0.2500000000000000</td>
<td>0.99048995710749580</td>
</tr>
<tr>
<td>1157</td>
<td>Jack</td>
<td>Stone</td>
<td>0.1000000000000000</td>
<td>0.601863084183319918</td>
</tr>
<tr>
<td>1277</td>
<td>Scott</td>
<td>Wagner</td>
<td>0.8500000000000000</td>
<td>0.4559420922850786</td>
</tr>
<tr>
<td>3188</td>
<td>Shirley</td>
<td>Flores</td>
<td>0.4000000000000000</td>
<td>0.538816771536005140</td>
</tr>
<tr>
<td>3196</td>
<td>Andrew</td>
<td>Holmes</td>
<td>0.9000000000000000</td>
<td>0.183954046444834949</td>
</tr>
<tr>
<td>3430</td>
<td>Philip</td>
<td>Little</td>
<td>0.1000000000000000</td>
<td>0.735279557092379495</td>
</tr>
<tr>
<td>3522</td>
<td>Jerry</td>
<td>Ross</td>
<td>0.8000000000000000</td>
<td>0.671828883472214349</td>
</tr>
<tr>
<td>3892</td>
<td>Barbara</td>
<td>Flores</td>
<td>0.3500000000000000</td>
<td>0.892901007123556866</td>
</tr>
</tbody>
</table>
```
Normalizing Salary Data Using Z-score

The following example shows how you can normalize the salary_data table using the Z-score normalization method.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

```sql
=> SELECT NORMALIZE('normalized_z_salary_data', 'salary_data', 'current_salary, years_worked', 'zscore');
NORMALIZE
------------------------
Finished in 1 iteration

=> SELECT * FROM normalized_z_salary_data;

<table>
<thead>
<tr>
<th>employee_id</th>
<th>first_name</th>
<th>last_name</th>
<th>years_worked</th>
<th>current_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>189</td>
<td>Shawn</td>
<td>Moore</td>
<td>-0.524447274157005</td>
<td>-0.221841249770669</td>
</tr>
<tr>
<td>518</td>
<td>Earl</td>
<td>Shaw</td>
<td>-1.35743214416495</td>
<td>1.66854215981221</td>
</tr>
<tr>
<td>1126</td>
<td>Susan</td>
<td>Alexander</td>
<td>-0.857641222160185</td>
<td>1.41799393943946</td>
</tr>
<tr>
<td>1157</td>
<td>Jack</td>
<td>Stone</td>
<td>-1.35743214416495</td>
<td>0.350834283622416</td>
</tr>
<tr>
<td>1277</td>
<td>Scott</td>
<td>Wagner</td>
<td>-1.52402911816495</td>
<td>-0.15068522159045</td>
</tr>
<tr>
<td>3188</td>
<td>Shirley</td>
<td>Flores</td>
<td>-0.357850300155415</td>
<td>0.131812255991634</td>
</tr>
<tr>
<td>3196</td>
<td>Andrew</td>
<td>Holmes</td>
<td>1.30811943986048</td>
<td>-1.10097599783475</td>
</tr>
<tr>
<td>3430</td>
<td>Philip</td>
<td>Little</td>
<td>-1.35743214416495</td>
<td>0.814321286168547</td>
</tr>
<tr>
<td>3522</td>
<td>Jerry</td>
<td>Ross</td>
<td>0.974925491857304</td>
<td>0.593894513770248</td>
</tr>
<tr>
<td>3892</td>
<td>Barbara</td>
<td>Flores</td>
<td>-0.524447274157005</td>
<td>-1.4172930118583</td>
</tr>
</tbody>
</table>
```

Normalizing Salary Data Using Robust Z-score

The following example shows how you can normalize the salary_data table using the Z-score normalization method.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

```sql
=> SELECT NORMALIZE('normalized_robustz_salary_data', 'salary_data', 'current_salary, years_worked', 'robust_zscore');
```
NORMALIZE

------------------------------
Finished in 1 iteration
(1 row)

=> SELECT * FROM normalized_robustz_salary_data;

<table>
<thead>
<tr>
<th>employee_id</th>
<th>first_name</th>
<th>last_name</th>
<th>years_worked</th>
<th>current_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>189</td>
<td>Shawn</td>
<td>Moore</td>
<td>-0.484694455685957</td>
<td>-0.15893384965549914</td>
</tr>
<tr>
<td>518</td>
<td>Earl</td>
<td>Shaw</td>
<td>-1.079185215162552</td>
<td>1.317126172796275889</td>
</tr>
<tr>
<td>1126</td>
<td>Susan</td>
<td>Alexander</td>
<td>-0.674490759476595</td>
<td>1.126852528914384584</td>
</tr>
<tr>
<td>1157</td>
<td>Jack</td>
<td>Stone</td>
<td>-1.079185215162552</td>
<td>0.289689691751547422</td>
</tr>
<tr>
<td>1277</td>
<td>Scott</td>
<td>Wagner</td>
<td>-1.214083367057871</td>
<td>-0.187964200747705982</td>
</tr>
<tr>
<td>3188</td>
<td>Shirley</td>
<td>Flores</td>
<td>-0.269796303790638</td>
<td>0.117871818982746738</td>
</tr>
<tr>
<td>3196</td>
<td>Andrew</td>
<td>Holmes</td>
<td>1.079185215162552</td>
<td>-0.849222942006447161</td>
</tr>
<tr>
<td>3430</td>
<td>Philip</td>
<td>Little</td>
<td>-1.079185215162552</td>
<td>0.6532845947426481</td>
</tr>
<tr>
<td>3522</td>
<td>Jerry</td>
<td>Ross</td>
<td>0.809389911371914</td>
<td>0.480364995828913355</td>
</tr>
<tr>
<td>3892</td>
<td>Barbara</td>
<td>Flores</td>
<td>-0.484694455685957</td>
<td>-1.097366550974798397</td>
</tr>
<tr>
<td>3939</td>
<td>Anna</td>
<td>Walker</td>
<td>-0.944287063267233</td>
<td>0.414956177842775781</td>
</tr>
<tr>
<td>4165</td>
<td>Martha</td>
<td>Reyes</td>
<td>0.269796303790638</td>
<td>0.773947781782753329</td>
</tr>
<tr>
<td>4335</td>
<td>Phillip</td>
<td>Wright</td>
<td>-1.214083367057871</td>
<td>1.218843012657445647</td>
</tr>
<tr>
<td>4534</td>
<td>Roger</td>
<td>Harris</td>
<td>1.079185215162552</td>
<td>1.155185821164402608</td>
</tr>
<tr>
<td>4806</td>
<td>John</td>
<td>Robinson</td>
<td>0.809389911371914</td>
<td>-0.494320112657445647</td>
</tr>
<tr>
<td>4881</td>
<td>Kelly</td>
<td>Welch</td>
<td>0.134898151895319</td>
<td>-0.540778080820045933</td>
</tr>
<tr>
<td>4889</td>
<td>Jennifer</td>
<td>Arnold</td>
<td>1.214083367057871</td>
<td>-0.299762093576526556</td>
</tr>
<tr>
<td>5067</td>
<td>Martha</td>
<td>Parker</td>
<td>0.000000000000000</td>
<td>0.71991343857328239</td>
</tr>
<tr>
<td>5523</td>
<td>John</td>
<td>Martin</td>
<td>-0.269796303790638</td>
<td>-0.411248545269163826</td>
</tr>
<tr>
<td>6004</td>
<td>Nicole</td>
<td>Sullivan</td>
<td>0.269796303790638</td>
<td>1.065141844522487821</td>
</tr>
<tr>
<td>6013</td>
<td>Harry</td>
<td>Woods</td>
<td>-0.944287063267233</td>
<td>1.00566438654129376</td>
</tr>
<tr>
<td>6240</td>
<td>Norma</td>
<td>Martinez</td>
<td>1.214083367057871</td>
<td>0.76241284488701691</td>
</tr>
</tbody>
</table>

See Also

- **NORMALIZE**

Sampling Data

The goal of data sampling is to take a smaller, more manageable sample of a much larger data set. With a sample data set, you can produce predictive models or use it to help you tune your database. The following example shows how you can use the TABLESAMPLE clause to create a sample of your data.
Sampling Data from a Table

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

Using the baseball table, create a new table named `baseball_sample` containing a 25% sample of baseball. Remember, TABLESAMPLE does not guarantee that the exact percentage of records defined in the clause are returned.

```sql
=> CREATE TABLE baseball_sample AS SELECT * FROM baseball TABLESAMPLE(25);
CREATE TABLE
=> SELECT * FROM baseball_sample;

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>dob</th>
<th>team</th>
<th>hr</th>
<th>hits</th>
<th>avg</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Amanda</td>
<td>Turner</td>
<td>1997-12-22</td>
<td>Maroon</td>
<td>58</td>
<td>177</td>
<td>0.187</td>
<td>8047721</td>
</tr>
<tr>
<td>20</td>
<td>Jesse</td>
<td>Cooper</td>
<td>1983-04-13</td>
<td>Yellow</td>
<td>97</td>
<td>39</td>
<td>0.523</td>
<td>4252837</td>
</tr>
<tr>
<td>22</td>
<td>Randy</td>
<td>Peterson</td>
<td>1980-05-28</td>
<td>Orange</td>
<td>14</td>
<td>16</td>
<td>0.141</td>
<td>11827728.1</td>
</tr>
<tr>
<td>24</td>
<td>Carol</td>
<td>Harris</td>
<td>1991-04-02</td>
<td>Fuscia</td>
<td>96</td>
<td>12</td>
<td>0.456</td>
<td>40572253.6</td>
</tr>
<tr>
<td>32</td>
<td>Rose</td>
<td>Morrison</td>
<td>1977-07-26</td>
<td>Goldenrod</td>
<td>27</td>
<td>153</td>
<td>0.442</td>
<td>14510752.49</td>
</tr>
<tr>
<td>50</td>
<td>Helen</td>
<td>Medina</td>
<td>1987-12-26</td>
<td>Maroon</td>
<td>12</td>
<td>150</td>
<td>0.54</td>
<td>32169267.91</td>
</tr>
<tr>
<td>70</td>
<td>Richard</td>
<td>Gilbert</td>
<td>1983-07-13</td>
<td>Khaki</td>
<td>1</td>
<td>250</td>
<td>0.213</td>
<td>40518422.76</td>
</tr>
<tr>
<td>81</td>
<td>Angela</td>
<td>Cole</td>
<td>1991-08-16</td>
<td>Violet</td>
<td>87</td>
<td>136</td>
<td>0.796</td>
<td>42875181.51</td>
</tr>
<tr>
<td>82</td>
<td>Elizabeth</td>
<td>Foster</td>
<td>1994-04-30</td>
<td>Indigo</td>
<td>46</td>
<td>163</td>
<td>0.481</td>
<td>33896975.53</td>
</tr>
<tr>
<td>98</td>
<td>Philip</td>
<td>Gardner</td>
<td>1992-05-06</td>
<td>Puce</td>
<td>39</td>
<td>239</td>
<td>0.697</td>
<td>20967488.67</td>
</tr>
<tr>
<td>102</td>
<td>Ernest</td>
<td>Freeman</td>
<td>1983-10-05</td>
<td>Turquoise</td>
<td>46</td>
<td>77</td>
<td>0.564</td>
<td>21444463.92</td>
</tr>
</tbody>
</table>
```

With your sample you can create a predictive model, or tune your database.

See Also

- [FROM Clause](#) (for more information about the TABLESAMPLE clause)
- [Building a Linear Regression Model](#)
- [Building a Logistic Regression Model](#)
- [Clustering Data Using k-means](#)
Regression Algorithms

Regression is an important and popular machine learning tool that makes predictions from data by learning the relationship between some features of the data and an observed value response. Regression is used to make predictions about profits, sales, temperature, stocks, and more. For example, you could use regression to predict the price of a house based on the location, the square footage, the size of the lot, and so on. In this example, the house's value is the outcome, and the other factors, such as location, are the features.

The optimal set of coefficients found for the regression's equation is known as the model. The relationship between the outcome and the features is summarized in the model, which can then be applied to different data sets, where the outcome value is unknown. The data in regression projects is divided into two data sets - one that is used to build the model, and one that is used to test the model.

Vertica supports three algorithms for regression:

- **Linear Regression**
- **Random Forest for Regression**
- **SVM (Support Vector Machine) for Regression**

Linear Regression

Using linear regression, you can model the linear relationship between independent variables, or features, and a dependent variable, or outcome. You can build linear regression models to:

- Fit a predictive model to a training data set of independent variables and some dependent variable. Doing so allows you to use feature variable values to make predictions on outcomes. For example, you can predict the amount of rain that will fall on a particular day of the year.

- Determine the strength of the relationship between an independent variable and some outcome variable. For example, suppose you want to determine the importance of various weather variables on the outcome of how much rain will fall. You can build a linear regression model based on observations of weather patterns and rainfall to find the answer.
Unlike Logistic Regression, which you use to determine a binary classification outcome, linear regression is primarily used to predict continuous numerical outcomes in linear relationships.

You can use the following functions to build a linear regression model, view the model, and use the model to make predictions on a set of test data:

- `LINEAR_REG`
- `PREDICT_LINEAR_REG`
- `GET_MODEL_SUMMARY`

For a complete programming example of how to use linear regression on a table in Vertica, see Building a Linear Regression Model.

**Building a Linear Regression Model**

This linear regression example uses a small data set set named faithful. The example shows how you can build a model to predict the value of eruptions, given the value of the waiting feature.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Create the linear regression model, named `linear_reg_faithful`, using the `faithful_training` training data:

   ```sql
   => SELECT LINEAR_REG('linear_reg_faithful', 'faithful_training', 'eruptions', 'waiting'
   USING PARAMETERS optimizer='BFGS');
   LINEAR_REG
   -----------------------------
   Finished in 6 iterations
   (1 row)
   ```

2. View the summary output of `linear_reg_faithful`:

   ```sql
   => SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='linear_reg_faithful');
   -----------
   details
   predictor|coefficient|std_err |t_value |p_value
   -----------------------------
   Intercept| -2.06795 | 0.21063| -9.81782| 0.00000
   waiting   | 0.07876  | 0.00292|26.96925 | 0.00000
   ```
3. Create a new table that contains the response values from running the PREDICT_LINEAR_REG function on your test data. Name this table pred_faithful_results:

```sql
=> CREATE TABLE pred_faithful_results AS
(SELECT id, eruptions, PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_reg_faithful')
AS pred
FROM faithful_testing);
```

4. View the results in the pred_faithful_results table:

```sql
=> SELECT * FROM pred_faithful_results ORDER BY id;

<table>
<thead>
<tr>
<th>id</th>
<th>eruptions</th>
<th>pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2.283</td>
<td>2.8151271587036</td>
</tr>
<tr>
<td>5</td>
<td>4.533</td>
<td>4.62659045686076</td>
</tr>
<tr>
<td>8</td>
<td>3.6</td>
<td>4.62659045686076</td>
</tr>
<tr>
<td>9</td>
<td>1.95</td>
<td>1.94877514654148</td>
</tr>
<tr>
<td>11</td>
<td>1.833</td>
<td>2.18585296804024</td>
</tr>
<tr>
<td>12</td>
<td>3.917</td>
<td>4.54783118302784</td>
</tr>
<tr>
<td>14</td>
<td>1.75</td>
<td>1.6337380512098</td>
</tr>
<tr>
<td>20</td>
<td>4.25</td>
<td>4.15483481386324</td>
</tr>
<tr>
<td>22</td>
<td>1.75</td>
<td>1.6337380512098</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(110 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Calculating the Mean Squared Error (MSE)

Another way that you can calculate how well your model fits the data is by using the MSE function. MSE returns the average of the squared differences between actual value and predicted values.

```
=> SELECT MSE (eruptions::float, pred::float) OVER() FROM
(SELECT eruptions, pred FROM pred_faithful_results) AS prediction_output;

<table>
<thead>
<tr>
<th>mse</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.252925741352641</td>
<td>Of 110 rows, 110 were used and 0 were ignored (1 row)</td>
</tr>
</tbody>
</table>
```

See Also

- LINEAR_REG
- PREDICT_LINEAR_REG
- GET_MODEL_SUMMARY
- RSQUARED
- MSE

Random Forest for Regression

The Random Forest for regression algorithm creates an ensemble model of regression trees. Each tree is trained on a randomly selected subset of the training data. The algorithm predicts the value that is the mean prediction of the individual trees.

You can use the following functions to train the Random Forest model, and use the model to make predictions on a set of test data:

- GET_MODEL_SUMMARY
- PREDICT_RF_REGRESSOR
- RF_REGRESSOR

For a complete example of how to use the Random Forest for regression algorithm in Vertica, see Building a Random Forest for Regression Model.
Building a Random Forest for Regression Model

This random forest example uses a small data set named mtcars. The example shows you how to build a model to predict the value of carb (the number of carburetors). It uses the given values of all the other features in the data set.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Create the random forest model, named myRFRegressorModel, using the mtcars_train training data.

```sql
=> SELECT RF_REGRESSOR ('myRFRegressorModel', 'mtcars', 'carb', 'mpg, cyl, hp, drat, wt' USING PARAMETERS ntree=100, sampling_size=0.3);
RF_REGRESSOR
-----------------
Finished
(1 row)
```

2. View the summary output of myRFRegressorModel.

```sql
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='myRFRegressorModel');
-----------------------------
call_string
-----------------------------
SELECT rf_regressor('public.myRFRegressorModel', 'mtcars', '"carb"', 'mpg, cyl, hp, drat, wt' USING PARAMETERS exclude_columns='', ntree=100, mtry=1, sampling_size=0.3, max_depth=5, max_breadth=32, min_leaf_size=5, min_info_gain=0, nbins=32);
-----------------------------
details
-----------------------------
predictor|type
---------|-----
mpg      |float
\_\_\_    |\_\_\_    
cyl      |int
\_\_\_    |\_\_\_    
hp       |int
\_\_\_    |\_\_\_    
drat     |float
\_\_\_    |\_\_\_    
wt       |float
\_\_\_    |\_\_\_    
-----------------------------
Additional Info
-----------------------------
Name     |Value
---------|-----
tree_count |100
rejected_row_count |0
```
3. Use PREDICT_RF_REGRESSOR to view the probability of the classes:

```sql
=> SELECT PREDICT_RF_REGRESSOR (mpg,cyl,hp,drat,wt
USING PARAMETERS model_name='myRFRegressorModel')FROM mtcars;
```

```plaintext
2.94774203574204
2.6954087024087
2.6954087024087
2.89906346431346
2.97688489288489
2.97688489288489
2.7086587024087
2.92078965478965
2.97688489288489
2.7086587024087
2.95621822621823
2.82255155955156
2.7086587024087
2.92255155955156
2.7086587024087
2.85650394050394
2.85650394050394
2.97688489288489
2.95621822621823
2.6954087024087
2.6954087024087
2.84493251193251
2.97688489288489
2.97688489288489
2.88564679764679
2.6954087024087
2.92078965478965
2.97688489288489
2.97688489288489
2.7934087024087
2.7934087024087
2.7086587024087
2.72469441669442
```

(32 rows)

See Also

- GET_MODEL_SUMMARY
- PREDICT_RF_REGRESSOR
- RF_REGRESSOR
SVM (Support Vector Machine) for Regression

Support Vector Machine (SVM) for regression predicts continuous ordered variables based on the training data.

Unlike Logistic Regression, which you use to determine a binary classification outcome, SVM for regression is primarily used to predict continuous numerical outcomes.

You can use the following functions to build an SVM for regression model, view the model, and use the model to make predictions on a set of test data:

- `SVM_REGRESSOR`
- `PREDICT_SVM_REGRESSOR`
- `GET_MODEL_SUMMARY`

For a complete example of how to use the SVM algorithm in Vertica, see Building an SVM for Regression Model.

Building an SVM for Regression Model

This SVM for regression example uses a small data set named faithful, based on the Old Faithful geyser in Yellowstone National Park. The data set contains values about the waiting time between eruptions and the duration of eruptions of the geyser. The example shows how you can build a model to predict the value of eruptions, given the value of the waiting feature.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Create the SVM model, named `svm_faithful`, using the `faithful_training` training data:

```sql
=> SELECT SVM_REGRESSOR('svm_faithful', 'faithful_training', 'eruptions', 'waiting'
    USING PARAMETERS error_tolerance=0.1, max_iterations=100);
```

```
SVM_REGRESSOR
***************
Finished in 5 iterations
Accepted Rows: 162  Rejected Rows: 0
(1 row)
```
2. View the summary output of `svm_faithful`:

```sql
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='svm_faithful');

Details

Predictors and Coefficients

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>-1.59007</td>
</tr>
<tr>
<td>waiting</td>
<td>0.07217</td>
</tr>
</tbody>
</table>

Call string:

```
SELECT svm_regressor('public.svm_faithful', 'faithful_training', '"eruptions"',
'waiting'USING PARAMETERS error_tolerance = 0.1, C=1, max_iterations=100,
epsilon=0.001);
```

Additional Info

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>accepted_row_count</td>
<td>162</td>
</tr>
<tr>
<td>rejected_row_count</td>
<td>0</td>
</tr>
<tr>
<td>iteration_count</td>
<td>5</td>
</tr>
</tbody>
</table>
```
```

(1 row)
```

3. Create a new table that contains the response values from running the PREDICT_SVM_REGRESSOR function on your test data. Name this table `pred_faithful_results`:

```sql
=> CREATE TABLE pred_faithful AS
(SELECT id, eruptions, PREDICT_SVM_REGRESSOR(waiting USING PARAMETERS model_name='svm_faithful')
AS pred
FROM faithful_testing);
```

4. View the results in the `pred_faithful_results` table:

```sql
=> SELECT * FROM pred_faithful ORDER BY id;
```

<table>
<thead>
<tr>
<th>id</th>
<th>eruptions</th>
<th>pred</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2.283</td>
<td>2.88444568755189</td>
</tr>
<tr>
<td>5</td>
<td>4.533</td>
<td>4.54434581879796</td>
</tr>
<tr>
<td>8</td>
<td>3.6</td>
<td>4.54434581879796</td>
</tr>
<tr>
<td>9</td>
<td>1.95</td>
<td>2.09058040739072</td>
</tr>
<tr>
<td>11</td>
<td>1.833</td>
<td>2.30788912016195</td>
</tr>
</tbody>
</table>
```
Calculating the Mean Squared Error (MSE)

Another way that you can calculate how well your model fits the data is by using the MSE function. MSE returns the average of the squared differences between actual value and predicted values.

```sql
=> SELECT MSE(obs::float, prediction::float) OVER()
    FROM (SELECT eruptions AS obs, pred AS prediction
          FROM pred_faithful) AS prediction_output;
```

<table>
<thead>
<tr>
<th>mse</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.254499811834235</td>
<td>Of 110 rows, 110 were used and 0 were ignored</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- **SVM (Support Vector Machine) for Regression**
- **SVM_REGRESSOR**
- **PREDICT_SVM_REGRESSOR**
- **GET_MODEL_SUMMARY**

## Classification Algorithms

Classification is an important and popular machine learning tool that assigns items in a data set to different categories. Classification is used to predict risk over time, in fraud detection, text categorization, and more. Classification functions begin with a data set where the different categories are known. For example, suppose you want to classify students based on how likely they are to get into graduate school. In addition to factors like admission score exams and grades, you could also track work experience.

Binary classification means the outcome, in this case, admission, only has two possible values: admit or do not admit. Multiclass outcomes have more than two values. For example, low,
medium, or high chance of admission. During the training process, classification algorithms find the relationship between the outcome and the features. This relationship is summarized in the model, which can then be applied to different data sets, where the categories are unknown. The data in classification projects is divided into two data sets - one that is used to build the model, and one that is used to test the model.

Vertica supports four algorithms for classification:

- Logistic Regression
- Naive Bayes
- Random Forest for Classification
- SVM (Support Vector Machine) for Classification

**Logistic Regression**

Using logistic regression, you can model the relationship between independent variables, or features, and some dependent variable, or outcome. The outcome of logistic regression is always a binary value.

You can build logistic regression models to:

- Fit a predictive model to a training data set of independent variables and some binary dependent variable. Doing so allows you to make predictions on outcomes, such as whether a piece of email is spam mail or not.

- Determine the strength of the relationship between an independent variable and some binary outcome variable. For example, suppose you want to determine whether an email is spam or not. You can build a logistic regression model, based on observations of the properties of email messages. Then, you can determine the importance of various properties of an email message on that outcome.

You can use the following functions to build a logistic regression model, view the model, and use the model to make predictions on a set of test data:

- LOGISTIC_REG
- PREDICT_LOGISTIC_REG
- GET_MODEL_SUMMARY
For a complete programming example of how to use logistic regression on a table in Vertica, see Building a Logistic Regression Model.

Building a Logistic Regression Model

This logistic regression example uses a small data set named mtcars. The example shows you how to build a model to predict the value of am (whether the car has an automatic or a manual transmission). It uses the given values of all the other features in the data set.

In this example, roughly 60% of the data is used as training data to create a model. The remaining 40% of the data is used as testing data against which you can test your logistic regression model.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Create the logistic regression model, named logistic_reg_mtcars, using the mtcars_train training data.

=> SELECT LOGISTIC_REG('logistic_reg_mtcars', 'mtcars_train', 'am', 'cyl, wt' USING PARAMETERS exclude_columns='hp');

<table>
<thead>
<tr>
<th>predictor</th>
<th>coefficient</th>
<th>std_err</th>
<th>z_value</th>
<th>p_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>262.39898</td>
<td>44745.77338</td>
<td>0.00586</td>
<td>0.99532</td>
</tr>
<tr>
<td>cyl</td>
<td>16.75892</td>
<td>5987.23236</td>
<td>0.00280</td>
<td>0.99777</td>
</tr>
<tr>
<td>wt</td>
<td>-119.92116</td>
<td>17237.03154</td>
<td>-0.00696</td>
<td>0.99445</td>
</tr>
</tbody>
</table>

2. View the summary output of logistic_reg_mtcars.

=> SELECT GET_MODEL_SUMMARY USING PARAMETERS model_name='logistic_reg_mtcars';
logistic_reg('public.logistic_reg_mtcars', 'mtcars_train', '"am"', 'cyl, wt'
USING PARAMETERS exclude_columns='hp', optimizer='newton', epsilon=1e-06,
max_iterations=100, regularization='none', lambda=1)

Using Parameters:
- exclude_columns='hp'
- optimizer='newton'
- epsilon=1e-06
- max_iterations=100
- regularization='none'
- lambda=1

3. Create a new table, named `mtcars_predict_results`. Populate this table with the prediction outputs you obtain from running the `PREDICT_LOGISTIC_REG` function on your test data.

```
=> CREATE TABLE mtcars_predict_results AS
  (SELECT car_model, am,
  PREDICT_LOGISTIC_REG(cyl, wt
  USING PARAMETERS model_name='logistic_reg_mtcars')
  AS Prediction
  FROM mtcars_test);
```

4. View the results in the `mtcars_predict_results` table.

```
=> SELECT * FROM mtcars_predict_results;

<table>
<thead>
<tr>
<th>car_model</th>
<th>am</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC Javelin</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maserati Bora</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Merc 280</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Toyota Corona</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Volvo 142E</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Camaro Z28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Porsche 914-2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Valiant</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(12 rows)
```

5. Evaluate the accuracy of the `PREDICT_LOGISTIC_REG` function, using the `CONFUSION_MATRIX` evaluation function.

```
=> SELECT CONFUSION_MATRIX(obs::int, pred::int USING PARAMETERS num_classes=2) OVER()
  FROM (SELECT am AS obs, Prediction AS pred FROM mtcars_predict_results) AS prediction_output;
```
In this case, PREDICT_LOGISTIC_REG correctly predicted that four out of five cars with a value of 1 in the am column have a value of 1. Out of the seven cars which had a value of 0 in the am column, six were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

See Also

- CONFUSION_MATRIX
- LIFT_TABLE
- LOGISTIC_REG
- PREDICT_LOGISTIC_REG
- GET_MODEL_SUMMARY

Naive Bayes

You can use the Naive Bayes algorithm to classify your data when features can be assumed independent. The algorithm uses independent features to calculate the probability of a specific class. For example, you might want to predict the probability that an email is spam. In that case, you would use a corpus of words associated with spam to calculate the probability the email's content is spam.

You can use the following functions to build a Naive Bayes model, view the model, and use the model to make predictions on a set of test data:

- NAIVE_BAYES
- PREDICT_NAIVE_BAYES
- PREDICT_NAIVE_BAYES_CLASSES
- GET_MODEL_SUMMARY

For a complete example of how to use the Naive Bayes algorithm in Vertica, see Classifying Data Using Naive Bayes.
Classifying Data Using Naive Bayes

This Naive Bayes example uses the HouseVotes84 data set to show you how to build a model. With this model, you can predict which party the member of the United States Congress is affiliated based on their voting record. To aid in classifying the data it has been cleaned, and any missed votes have been replaced. The cleaned data replaces missed votes with the voter's party majority vote. For example, suppose a member of the Democrats had a missing value for vote1 and majority of the Democrats voted in favor. This example replaces all missing Democrats' votes for vote1 with a vote in favor.

In this example, approximately 75% of the cleaned HouseVotes84 data is randomly selected and copied to a training table. The remaining cleaned HouseVotes84 data is used as a testing table.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

1. Create the Naive Bayes model, named naive_house84_model, using the house84_train training data.

```sql
=> SELECT NAIVE_BAYES('naive_house84_model', 'house84_train', 'party',
                          '*' USING PARAMETERS exclude_columns='party, id');

NAIVE_BAYES
-------------------------
Finished. Accepted Rows: 315  Rejected Rows: 0
(1 row)
```

2. Create a new table, named predicted_party_naive. Populate this table with the prediction outputs you obtain from the PREDICT_NAIVE_BAYES function on your test data.

```sql
=> CREATE TABLE predicted_party_naive
AS SELECT party,
         PREDICT_NAIVE_BAYES (vote1, vote2, vote3, vote4, vote5,
                                vote6, vote7, vote8, vote9, vote10,
                                vote11, vote12, vote13, vote14,
                                vote15, vote16
                        USING PARAMETERS model_name = 'naive_house84_model',
                              type = 'response') AS Predicted_Party
FROM house84_test;
```

3. Calculate the accuracy of the model's predictions.

```sql
=> SELECT (Predictions.Num_Correct_Predictions / Count.Total_Count) AS Percent_Accuracy
FROM (SELECT COUNT(Predicted_Party) AS Num_Correct_Predictions
       FROM predicted_party_naive)
```
WHERE party = Predicted_Party
) AS Predictions,
(SELECT COUNT(party) AS Total_Count
FROM predicted_party_naive
) AS Count;

Percent_Accuracy
------------------------
0.933333333333333333
(1 row)

The model correctly predicted the party of the members of Congress based on their voting patterns with 93% accuracy.

Viewing the Probability of Each Class

You can also view the probability of each class. Use PREDICT_NAIVE_BAYES_CLASSES to see the probability of each class.


<table>
<thead>
<tr>
<th>id</th>
<th>Predicted</th>
<th>Probability</th>
<th>democrat</th>
<th>republican</th>
</tr>
</thead>
<tbody>
<tr>
<td>368</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>372</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>374</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>378</td>
<td>republican</td>
<td>0.99999962214987</td>
<td>3.77850125111219e-08</td>
<td>0.99999962214987</td>
</tr>
<tr>
<td>384</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>387</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>406</td>
<td>republican</td>
<td>0.99999945980143</td>
<td>5.40198564592332e-08</td>
<td>0.99999945980143</td>
</tr>
<tr>
<td>419</td>
<td>democrat</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>421</td>
<td>republican</td>
<td>0.922808855631005</td>
<td>0.0771911443689949</td>
<td>0.922808855631005</td>
</tr>
</tbody>
</table>

(109 rows)

See Also

- NAIVE_BAYES
- PREDICT_NAIVE_BAYES
- PREDICT_NAIVE_BAYES_CLASSES
Random Forest for Classification

The Random Forest algorithm creates an ensemble model of decision trees. Each tree is trained on a randomly selected subset of the training data.

You can use the following functions to train the Random Forest model, and use the model to make predictions on a set of test data:

- RF_CLASSIFIER
- PREDICT_RF_CLASSIFIER
- PREDICT_RF_CLASSIFIER_CLASSES
- GET_MODEL_SUMMARY

For a complete example of how to use the Random Forest algorithm in Vertica, see Classifying Data Using Random Forest.

Classifying Data Using Random Forest

This random forest example uses a data set named iris. The example contains four variables that measure various parts of the iris flower to predict its species.

Before you begin the example, make sure that you have followed the steps in Downloading the Machine Learning Example Data.

1. Create the random forest model, named rf_iris, using the iris data.

   ```sql
   => SELECT RF_CLASSIFIER ('rf_iris', 'iris', 'Species', 'Sepal_Length, Sepal_Width, Petal_Length, Petal_Width')
   USING PARAMETERS ntree=100, sampling_size=0.5);
   
   RF_CLASSIFIER
   ---------------------------
   The random forest is trained
   
   (1 row)
   ```

2. View the summary output of rf_iris.

   ```sql
   ```
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='rf_iris');

---------
call_string
---------
SELECT rf_classifier('public.rf_iris', 'iris', '"species"', 'Sepal_Length, Sepal_Width, Petal_Length, Petal_Width' USING PARAMETERS exclude_columns='', ntree=100, mtry=2, sampling_size=0.5, max_depth=5, max_breadth=32, min_leaf_size=1, min_info_gain=0, nbins=32);

---------
details
---------
predictor | type
-----------
sepal_length | float
sepal_width | float
petal_length | float
petal_width | float

---------
Additional Info
---------
Name | Value
----- | ----
tree_count | 100
rejected_row_count | 0
accepted_row_count | 150
(1 row)

3. Apply the classifier to the test data:

=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
USING PARAMETERS model_name='rf_iris') FROM iris1;

PREDICT_RF_CLASSIFIER
---------------------
setosa
setosa
setosa
.
.
versicolor
versicolor
versicolor
.
.
virginica
virginica
virginica
.
.
4. Use PREDICT_RF_CLASSES to view the probability of the classes:

```sql
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
   USING PARAMETERS model_name='rf_iris') OVER () FROM iris1;
```

<table>
<thead>
<tr>
<th>predicted</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>0.99</td>
</tr>
</tbody>
</table>

See Also

- RF_CLASSIFIER
- PREDICT_RF_CLASSIFIER
- PREDICT_RF_CLASSIFIER_CLASSES

**SVM (Support Vector Machine) for Classification**

Support Vector Machine (SVM) is a classification algorithm that assigns data to one category or the other based on the training data. This algorithm implements linear SVM, which is highly scalable.

You can use the following functions to train the SVM model, and use the model to make predictions on a set of test data:

- SVM_CLASSIFIER
- PREDICT_SVM_CLASSIFIER
You can also use the following evaluation functions to gain further insights:

- **CONFUSION_MATRIX**
- **DETECT_OUTLIERS**
- **ERROR_RATE**
- **ROC**

For a complete example of how to use the SVM algorithm in Vertica, see [Classifying Data Using SVM (Support Vector Machine)](Vertica Documentation Analyzing Data)

The implementation of the SVM algorithm in Vertica is based on the paper [Distributed Newton Methods for Regularized Logistic Regression](Vertica Documentation Analyzing Data).

**Classifying Data Using SVM (Support Vector Machine)**

This SVM example uses a small data set named mtcars. The example shows how you can use the SVM_CLASSIFIER function to train the model to predict the value of \( a_\text{m} \) (the transmission type, where 0 = automatic and 1 = manual) using the PREDICT_SVM_CLASSIFIER function.

Before you begin the example, make sure that you have [Downloading the Machine Learning Example Data](Vertica Documentation Analyzing Data).

1. Create the SVM model, named \texttt{svm\_class}, using the \texttt{mtcars\_train} training data.

   ```sql
   => SELECT SVM_CLASSIFIER('svm_class', 'mtcars_train', 'am', 'cyl, mpg, wt, hp, gear'
   USING PARAMETERS exclude_columns='gear');
   SVM_CLASSIFIER
   Finished in 12 iterations.
   Accepted Rows: 20  Rejected Rows: 0
   (1 row)
   ```

2. View the summary output of \texttt{svm\_class}.

   ```sql
   => SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='svm_class');
   ---------------------------
   details
   ```
<table>
<thead>
<tr>
<th>predictor</th>
<th>coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>-0.02006</td>
</tr>
<tr>
<td>cyl</td>
<td>0.15367</td>
</tr>
<tr>
<td>mpg</td>
<td>0.15698</td>
</tr>
<tr>
<td>wt</td>
<td>-1.78157</td>
</tr>
<tr>
<td>hp</td>
<td>0.00957</td>
</tr>
</tbody>
</table>

```
call_string
SELECT svm_classifier('public.svm_class', 'mtcars_train', 'am', 'cyl, mpg, wt, hp, gear' USING PARAMETERS exclude_columns='gear', C=1, max_iterations=100, epsilon=0.001);
```

```
--------------
Additional Info
--------------
Name           | Value
---------------+-----
accepted_row_count | 20
rejected_row_count | 0
iteration_count   | 12
(1 row)
```

3. **Create a new table, named svm_mtcars_predict.** Populate this table with the prediction outputs you obtain from running the PREDICT_SVM_CLASSIFIER function on your test data.

```sql
=> CREATE TABLE svm_mtcars_predict AS
    (SELECT car_model, am, PREDICT_SVM_CLASSIFIER(cyl, mpg, wt, hp
     USING PARAMETERS model_name='svm_class')
     AS Prediction FROM mtcars_test);
```

4. **View the results in the svm_mtcars_predict table.**

```sql
=> SELECT * FROM svm_mtcars_predict;
car_model         | am | Prediction
------------------+----|-----------
Toyota Corona     | 0  | 1         
Camaro Z28        | 0  | 0         
Datsun 710        | 1  | 1         
Valiant           | 0  | 0         
Volvo 142E        | 1  | 1         
AMC Javelin       | 0  | 0         
Honda Civic       | 1  | 1         
Hornet 4 Drive    | 0  | 0         
Maserati Bora     | 1  | 1         
Merc 280          | 0  | 0         
Merc 450SL        | 0  | 0         
```
5. Evaluate the accuracy of the PREDICT_SVM_CLASSIFIER function, using the CONFUSION_MATRIX evaluation function.

```sql
=> SELECT CONFUSION_MATRIX(obs::int, pred::int USING PARAMETERS num_classes=2) OVER()
    FROM (SELECT am AS obs, Prediction AS pred FROM svm_mtcars_predict1) AS prediction_output;
```

<table>
<thead>
<tr>
<th>class</th>
<th>0</th>
<th>1</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>Of 12 rows, 12 were used and 0 were ignored</td>
</tr>
</tbody>
</table>

In this case, PREDICT_SVM_CLASSIFIER correctly predicted that the cars with a value of 1 in the am column have a value of 1. No cars were incorrectly classified. Out of the seven cars which had a value of 0 in the am column, six were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

See Also

- SVM (Support Vector Machine) for Classification
- SVM_CLASSIFIER
- PREDICT_SVM_CLASSIFIER

## Clustering Algorithms

Clustering is an important and popular machine learning tool used to find clusters of items in a data set that are similar to one another. The goal of clustering is to create clusters with a high number of objects that are similar. Similar to classification, clustering segments the data. However, in clustering, the categorical groups are not defined. Clustering can be used to find anomalies in data and find natural groups of data. For example, you can use clustering to analyze a region and determine which areas of that region are most likely to be hit by an earthquake.

In Vertica, clustering is computed based on distance. Through this computation, data points are assigned to the cluster with the nearest mean.

Vertica supports one algorithm for clustering:
k-means

You can use the clustering algorithm, *k-means clustering*, to cluster data points into *k* different groups based on similarities between the data points.

The purpose of k-means is to partition *n* observations into *k* clusters. Through this partitioning, k-means assigns each observation to the cluster with the nearest mean. That nearest mean is also known as the *cluster center*.

For a complete programming example of how to use k-means on a table in Vertica, see *Clustering Data Using k-means*.

**Clustering Data Using k-means**

This k-means example uses two small data sets named *agar_dish_1* and *agar_dish_2*. Using the numeric data in the *agar_dish_1* data set, you can cluster the data into *k* clusters. Then, using the created k-means model, you can run APPLY_KMEANS on *agar_dish_2* and assign them to the clusters created in your original model.

Before you begin the example, make sure that you have loaded the Machine Learning sample data.

**Clustering Training Data into k Clusters**

1. Using the KMEANS function, run k-means on the *agar_dish_1* table.

   ```sql
   => SELECT KMEANS('agar_dish_kmeans', 'agar_dish_1', '*', 5
   USING PARAMETERS exclude_columns='id', max_iterations=20, output_view=agar_1_view,
   key_columns='id');
   ``

   The example creates a model named *agar_dish_kmeans* and a view containing the results of the model named *agar_1_view*. You might get different results when you run the clustering algorithm. This is because KMEANS randomly picks initial centers by default.
2. View the output of agar_1_view.

```sql
=> SELECT * FROM agar_1_view;

<table>
<thead>
<tr>
<th>id</th>
<th>cluster_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
</tr>
</tbody>
</table>

(375 rows)
```

3. Because you specified the number of clusters as 5, verify that the function created five clusters. Count the number of data points within each cluster.

```sql
=> SELECT cluster_id, COUNT(cluster_id) as Total_count
   FROM agar_1_view
   GROUP BY cluster_id;

<table>
<thead>
<tr>
<th>cluster_id</th>
<th>Total_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>74</td>
</tr>
<tr>
<td>3</td>
<td>73</td>
</tr>
<tr>
<td>4</td>
<td>72</td>
</tr>
</tbody>
</table>

(5 rows)
```

From the output, you can see that five clusters were created: 0, 1, 2, 3, and 4.

You have now successfully clustered the data from agar_dish_1.csv into five distinct clusters.

Summarizing Your Model

You can also view a summary of the model you created using the `GET_MODEL_SUMMARY` function. This summary tells you how many cluster centers your model contains, along with other metrics.

```sql
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='agar_dish_kmeans');
```

```
<table>
<thead>
<tr>
<th>centers</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>0.49708</td>
</tr>
<tr>
<td>-7.48119</td>
</tr>
<tr>
<td>-1.56238</td>
</tr>
</tbody>
</table>
```
Clustering Data Using a k-means Model

Using agar_dish_kmeans, the k-means model you just created, you can classify the agar_dish_2 data set.

Create a table named kmeans_results, using the agar_dish_2 table as your input table and the agar_dish_kmeans model for your initial cluster centers.

Add only the relevant feature columns to the arguments in the APPLY_KMEANS function.

```sql
=> CREATE TABLE kmeans_results AS
  (SELECT id,
   APPLY_KMEANS(x, y
   USING PARAMETERS model_name='agar_dish_kmeans') AS cluster_id
  FROM agar_dish_2);
```

The kmeans_results table shows that the agar_dish_kmeans model correctly clustered the agar_dish_2 data.

See Also

- APPLY_KMEANS
- KMEANS
Model Management

The following topics explain how to manage your models:

- Altering Models
- Drop Models
- Managing Model Security
- Model Attributes
- Summarizing Models
- Viewing Models

As of 8.0.x, Vertica supports upgrading models.

Altering Models

After you create a model, you can alter it in three ways: renaming the model, changing the owner, and changing the schema.

The following example shows how you can use ALTER_MODEL to rename a model:

1. Find the model you want to alter.

   ```sql
   => SELECT * FROM V_CATALOG.MODELS WHERE model_name='mymodel';
   - [ RECORD 1 ]-+----------------------------------+
   | model_id | 45035996273816618 |
   | model_name | mymodel |
   | schema_id | 45035996273704978 |
   | schema_name | public |
   | owner_id | 45035996273704962 |
   | owner_name | dbadmin |
   | category | VERTICA_MODELS |
   | model_type | kmeans |
   | is_complete | t |
   | create_time | 2017-03-02 11:16:04.990626-05 |
   | size | 964 |
   ```

2. Rename the model.
ALTER MODEL mymodel RENAME TO mykmeansmodel;
ALTER MODEL

3. Review V_CATALOG.MODELS to verify that the model name was changed.

```sql
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';
```

<table>
<thead>
<tr>
<th>record</th>
<th>-------------------------------------------------------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_id</td>
<td>45035996273816618</td>
</tr>
<tr>
<td>model_name</td>
<td>mykmeansmodel</td>
</tr>
<tr>
<td>schema_id</td>
<td>45035996273704978</td>
</tr>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>owner_id</td>
<td>45035996273704962</td>
</tr>
<tr>
<td>owner_name</td>
<td>dbadmin</td>
</tr>
<tr>
<td>category</td>
<td>VERTICA_MODELS</td>
</tr>
<tr>
<td>model_type</td>
<td>kmeans</td>
</tr>
<tr>
<td>is_complete</td>
<td>t</td>
</tr>
<tr>
<td>create_time</td>
<td>2017-03-02 11:16:04.990626-05</td>
</tr>
<tr>
<td>size</td>
<td>964</td>
</tr>
</tbody>
</table>

The following example shows how you can use ALTER_MODEL to change the model owner:

1. Find the model you want to alter.

```sql
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';
```

<table>
<thead>
<tr>
<th>record</th>
<th>-------------------------------------------------------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_id</td>
<td>45035996273816618</td>
</tr>
<tr>
<td>model_name</td>
<td>mykmeansmodel</td>
</tr>
<tr>
<td>schema_id</td>
<td>45035996273704978</td>
</tr>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>owner_id</td>
<td>45035996273704962</td>
</tr>
<tr>
<td>owner_name</td>
<td>dbadmin</td>
</tr>
<tr>
<td>category</td>
<td>VERTICA_MODELS</td>
</tr>
<tr>
<td>model_type</td>
<td>kmeans</td>
</tr>
<tr>
<td>is_complete</td>
<td>t</td>
</tr>
<tr>
<td>create_time</td>
<td>2017-03-02 11:16:04.990626-05</td>
</tr>
<tr>
<td>size</td>
<td>964</td>
</tr>
</tbody>
</table>

2. Change the model owner.

```sql
=> ALTER MODEL mykmeansmodel OWNER TO user1;
ALTER MODEL
```

3. Review V_CATALOG.MODELS to verify that the owner was changed.

```sql
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';
```

<table>
<thead>
<tr>
<th>record</th>
<th>-------------------------------------------------------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_id</td>
<td>45035996273816618</td>
</tr>
<tr>
<td>model_name</td>
<td>mykmeansmodel</td>
</tr>
<tr>
<td>schema_id</td>
<td>45035996273704978</td>
</tr>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
</tbody>
</table>
The following example shows how you can use ALTER_MODEL to change the model schema:

1. Find the model you want to alter.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';
-[ RECORD 1 ]----------------------------------------------------------
model_id    | 45035996273816618
model_name  | mykmeansmodel
schema_id   | 45035996273704978
schema_name | public
owner_id    | 45035996273704962
owner_name  | dbadmin
category    | VERTICA_MODELS
model_type   | kmeans
is_complete | t
create_time | 2017-03-02 11:16:04.990626-05
size         | 964
```

2. Change the model schema.

```
=> ALTER MODEL mykmeansmodel SET SCHEMA test;
ALTER MODEL
```

3. Review V_CATALOG.MODELS to verify that the owner was changed.

```
=> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mykmeansmodel';
-[ RECORD 1 ]----------------------------------------------------------
model_id    | 45035996273816618
model_name  | mykmeansmodel
schema_id   | 45035996273704978
schema_name | test
owner_id    | 45035996273704962
owner_name  | dbadmin
category    | VERTICA_MODELS
model_type   | kmeans
is_complete | t
create_time | 2017-03-02 11:16:04.990626-05
size         | 964
```

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.
See Also

- ALTER MODEL

Drop Models

If you want to remove a model from the database, then you can drop it. To drop a model you must either be the model owner or the dbadmin.

1. Find the model you want to drop.

```sql
>> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mySvmClassModel';
-[ RECORD 1 ]----------------------------------------
  model_id | 45035996273765414
  model_name | mySvmClassModel
  schema_id | 45035996273704978
  schema_name | public
  owner_id | 45035996273704962
  owner_name | dbadmin
  category | VERTICA_MODELS
  model_type | SVM_CLASSIFIER
  is_complete | t
  create_time | 2017-02-14 10:30:44.903946-05
  size | 525
```

2. Drop the model.

```sql
>> DROP MODEL mySvmClassModel;
  DROP MODEL
```

3. Review V_CATALOG.MODELS to verify that the model was dropped.

```sql
>> SELECT * FROM V_CATALOG.MODELS WHERE model_name='mySvmClassModel';
(0 rows)
```

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

See Also

- DROP MODEL
Managing Model Security

You can manage the security privileges on your models by using the GRANT and REVOKE statements. The following examples show how you can change privileges on user1 and user2 using the faithful table using the linearReg model.

- In the following example, the dbadmin grants the SELECT privilege to user1:

  ```sql
  => GRANT SELECT ON TABLE faithful TO user1;
  GRANT PRIVILEGE
  ```

- Then, the dbadmin grants the CREATE privilege on the public schema to user1:

  ```sql
  => GRANT CREATE ON SCHEMA public TO user1;
  GRANT PRIVILEGE
  ```

- Connect to the database as user1:

  ```sql
  => \c - user1
  ```

- As user1, build the linearReg model:

  ```sql
  => SELECT LINEAR_REG('linearReg', 'faithful', 'waiting', 'eruptions');
  LINEAR_REG
  ------------------------
  Finished in 1 iterations
  (1 row)
  ```

- As user1, grant USAGE privileges to user2:

  ```sql
  => GRANT USAGE ON MODEL linearReg TO user2;
  GRANT PRIVILEGE
  ```

- Connect to the database as user2:

  ```sql
  => \c - user2
  ```

- To confirm privileges were granted to user2, run the GET_MODEL_SUMMARY function. A user with the USAGE privilege on a model can run GET_MODEL_SUMMARY on that model:
SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='linearReg');

details

predictor|coefficient|std_err |t_value |p_value
---|---|---|---|---
Intercept| 33.47440 | 1.15487|28.98533|0.00000
eruptions| 10.72964 | 0.31475|34.08903|0.00000

regualration

type| lambda
---|---
none| 1.00000

call_string

linear_reg('public.linearReg', 'faithful', '"waiting", 'eruptions'
USING PARAMETERS optimizer='newton', epsilon=1e-06, max_iterations=100, regularization='none', lambda=1)

Additional Info

Name | Value
---|---
iteration_count | 1
rejected_row_count| 0
accepted_row_count | 272
(1 row)

- Connect to the database as user1:

  => \c - user1

- Then, you can use the REVOKE statement to revoke privileges from user2:

  => REVOKE USAGE ON MODEL linearReg FROM user2;
  REVOKE PRIVILEGE

- To confirm the privileges were revoked, connect as user 2 and run the GET_MODEL_ SUMMARY function:

  => \c - user2
  =>SELECT GET_MODEL_SUMMARY('linearReg');
  ERROR 7523: Problem in get_model_summary.
  Detail: Permission denied for model linearReg
See Also

- GRANT (Model)
- REVOKE (Model)

Model Attributes

The following topics explain the model attributes for the Vertica machine learning algorithms. These attributes refer to the internal representation of the data generated by a particular model:

- Cross Validation Attributes
- K-means Model Attributes
- Naive Bayes Model Attributes
- Normalization Attributes
- One Hot Encoder Attributes
- Random Forest Model Attributes
- Regression Model Attributes
- SVM Model Attributes

Cross Validation Attributes

The following is a list of attributes from the `CROSS_VALIDATE` function:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_string</td>
<td>The value of all input arguments that were specified at the time the CROSS_VALIDATE function was called.</td>
</tr>
<tr>
<td>run_average</td>
<td>The average accuracy/error rate across all folds:</td>
</tr>
<tr>
<td>fold_info</td>
<td>The number of rows in each fold:</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td></td>
<td>• fold_id: The index of the fold.</td>
</tr>
<tr>
<td></td>
<td>• row_count: The number of rows held out for testing in the fold.</td>
</tr>
<tr>
<td>counters</td>
<td>All counters for the function including:</td>
</tr>
<tr>
<td></td>
<td>• accepted_row_count: The total number of rows in the input_relation, minus the number of rejected rows.</td>
</tr>
<tr>
<td></td>
<td>• rejected_row_count: The number of rows of the input_relation that were skipped because they contained an invalid value.</td>
</tr>
<tr>
<td></td>
<td>• feature_count: The number of features input to the machine learning model.</td>
</tr>
<tr>
<td>run_details</td>
<td>Information about each run, where a run means training a single model, and then testing that model on the one held-out fold:</td>
</tr>
<tr>
<td></td>
<td>• fold_id: The index of the fold held out for testing.</td>
</tr>
<tr>
<td></td>
<td>• iteration_count: The number of iterations used in model training on non-held-out folds.</td>
</tr>
<tr>
<td></td>
<td>• accuracy: One minus the value of the error_rate.</td>
</tr>
<tr>
<td></td>
<td>• error_rate: The average error rate across all folds.</td>
</tr>
</tbody>
</table>
K-means Model Attributes

The following is a list of attributes from the KMEANS function:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>centers</td>
<td>A list that contains the center of each cluster.</td>
</tr>
<tr>
<td>metrics</td>
<td>A string summary of several metrics related to the quality of the clustering.</td>
</tr>
</tbody>
</table>

Naive Bayes Model Attributes

The following is a list of attributes from the NAIVE_BAYES function:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colsInfo</td>
<td>The information from the response and predictor columns used in training:</td>
</tr>
<tr>
<td></td>
<td>- index: The index (starting at 0) of the column as provided in training. Index 0 is used for the response column.</td>
</tr>
<tr>
<td></td>
<td>- name: The column name.</td>
</tr>
<tr>
<td></td>
<td>- type: The label used for response with a value of Gaussian, Multinominal, Categorical, or Bernoulli.</td>
</tr>
<tr>
<td>alpha</td>
<td>The smooth parameter value.</td>
</tr>
<tr>
<td>prior</td>
<td>The percentage of each class among all training samples:</td>
</tr>
<tr>
<td></td>
<td>- label: The class label.</td>
</tr>
<tr>
<td></td>
<td>- value: The percentage of each class.</td>
</tr>
<tr>
<td>nRowsTotal</td>
<td>The number of samples accepted for training from the data set.</td>
</tr>
</tbody>
</table>
### nRowsRejected
The number of samples rejected for training.

### callStr
The SQL statement used to replicate the training.

### Gaussian
The Gaussian model conditioned on the class indicated by the class_name:
- **index**: The index of the predictor column.
- **mu**: The mean value of the model.
- **sigmaSq**: The squared standard deviation of the model.

### Multinominal
The Multinomial model conditioned on the class indicated by the class_name:
- **index**: The index of the predictor column.
- **prob**: The probability conditioned on the class indicated by the class_name.

### Bernoulli
The Bernoulli model conditioned on the class indicated by the class_name:
- **index**: The index of the predictor column.
- **probTrue**: The probability of having the value TRUE in this predictor column.

### Categorical
The Gaussian model conditioned on the class indicated by the class_name:
- **category**: The value in the predictor name.
- **<class_name>**: The probability of having that value conditioned on the class indicated by the class_name.

### Normalization Attributes
The following is a list of attributes from the `NORMALIZE_FIT` function for the minmax normalization method:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A summary of the data for the function including:</td>
</tr>
<tr>
<td></td>
<td>- colNames: The names of the columns in the model.</td>
</tr>
<tr>
<td></td>
<td>- mins: The minimum value of each column.</td>
</tr>
<tr>
<td></td>
<td>- maxes: The maximum value of each column.</td>
</tr>
</tbody>
</table>

The following is a list of attributes from the `NORMALIZE_FIT` function for the zscore normalization method:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A summary of the data for the function including:</td>
</tr>
<tr>
<td></td>
<td>- colNames: The names of the columns in the model.</td>
</tr>
<tr>
<td></td>
<td>- avgs: The average value of each column.</td>
</tr>
<tr>
<td></td>
<td>- stddev: The standard deviation of each column.</td>
</tr>
</tbody>
</table>

The following is a list of attributes from the `NORMALIZE_FIT` function for the robust_zscore normalization method:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A summary of the data for the function including:</td>
</tr>
<tr>
<td></td>
<td>- colNames: The names of the columns in the model.</td>
</tr>
<tr>
<td></td>
<td>- meds: The approximate median value of each column.</td>
</tr>
<tr>
<td></td>
<td>- mads: The median absolute deviation (MAD) from the median of each column.</td>
</tr>
</tbody>
</table>
## One Hot Encoder Attributes

The following is a list of attributes from the `ONE_HOT_ENCODER_FIT` function:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_string</td>
<td>The value of all input arguments that were specified at the time the function was called.</td>
</tr>
<tr>
<td>varchar_categories</td>
<td>The CHAR and VARCHAR categories:</td>
</tr>
<tr>
<td></td>
<td>• category_name: The name of the column.</td>
</tr>
<tr>
<td></td>
<td>• category_level: The levels of the category, in sorted order for each category.</td>
</tr>
<tr>
<td></td>
<td>• category_level_index: The index of this categorical level in the sorted list of levels for the category.</td>
</tr>
<tr>
<td>integer_categories</td>
<td>The INTEGER categories:</td>
</tr>
<tr>
<td></td>
<td>• category_name: The name of the column.</td>
</tr>
<tr>
<td></td>
<td>• category_level: The levels of the category, in sorted order for each category.</td>
</tr>
<tr>
<td></td>
<td>• category_level_index: The index of this categorical level in the sorted list of levels for the category.</td>
</tr>
<tr>
<td>boolean_categories</td>
<td>The BOOLEAN categories:</td>
</tr>
<tr>
<td></td>
<td>• category_name: The name of the column.</td>
</tr>
<tr>
<td></td>
<td>• category_level: The levels of the category, in sorted order for each category.</td>
</tr>
<tr>
<td></td>
<td>• category_level_index: The index of this categorical level in the sorted list of levels for the category.</td>
</tr>
<tr>
<td>date_categories</td>
<td>The DATE categories:</td>
</tr>
</tbody>
</table>

Random Forest Model Attributes

The following is a list of attributes for the random forest algorithms:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The data for the function, including:</td>
</tr>
<tr>
<td></td>
<td>• predictorNames: The name of the predictors in the same order they were specified for training the model.</td>
</tr>
<tr>
<td></td>
<td>• predictorTypes: The type of the predictors in the same order as their names in predictorNames.</td>
</tr>
<tr>
<td>ntree</td>
<td>The number of trees in the model.</td>
</tr>
<tr>
<td>skippedRows</td>
<td>The number of rows in input_relation that were skipped because they contained an invalid value.</td>
</tr>
<tr>
<td>processedRows</td>
<td>The total number of rows in input_relation minus skippedRows.</td>
</tr>
<tr>
<td>callStr</td>
<td>The value of all input arguments that were specified at the time the function was called.</td>
</tr>
</tbody>
</table>

Regression Model Attributes

The following is a list of attributes for the regression algorithms:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The data for the function, including:</td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td></td>
<td>• coeffNames: The name of the coefficients. This starts with intercept and then follows with the names of the predictors in the same order specified in the call.</td>
</tr>
<tr>
<td></td>
<td>• coeff: The vector of estimated coefficients, with the same order as coeffNames.</td>
</tr>
<tr>
<td></td>
<td>• stdErr: The vector of the standard error of the coefficients, with the same order as coeffNames.</td>
</tr>
<tr>
<td></td>
<td>• zValue (for logistic regression): The vector of z-values of the coefficients, in the same order as coeffNames.</td>
</tr>
<tr>
<td></td>
<td>• tValue (for linear regression): The vector of t-values of the coefficients, in the same order as coeffNames.</td>
</tr>
<tr>
<td></td>
<td>• pValue: The vector of p-values of the coefficients, in the same order as coeffNames.</td>
</tr>
<tr>
<td>regularization</td>
<td>The type of regularization to use when training the model.</td>
</tr>
<tr>
<td>lambda</td>
<td>The regularization parameter. Higher values enforce stronger regularization. This value must be positive.</td>
</tr>
<tr>
<td>alpha</td>
<td>The elastic net mixture parameter.</td>
</tr>
<tr>
<td>iterations</td>
<td>The number of iterations that actually occur for the convergence before exceeding max_iteration.</td>
</tr>
<tr>
<td>skippedRows</td>
<td>The number of rows of input_relation that were skipped because they contained an invalid value.</td>
</tr>
<tr>
<td>processedRows</td>
<td>The total number of rows in input_</td>
</tr>
</tbody>
</table>
relation minus the skippedRows.

callStr  
The value of all input arguments that were specified at the time the function was called.

**SVM Model Attributes**

The following is a list of attributes for the SVM algorithms:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff</td>
<td>The coefficients in the model:</td>
</tr>
<tr>
<td></td>
<td>• colNames: Either intercept or the name of the predictor column.</td>
</tr>
<tr>
<td></td>
<td>• coefficients: The value of the coefficient.</td>
</tr>
<tr>
<td>nAccepted</td>
<td>The number of samples accepted for training from the data set.</td>
</tr>
<tr>
<td>nRejected</td>
<td>The number of samples rejected when training.</td>
</tr>
<tr>
<td>nIteration</td>
<td>The number of iterations used in training.</td>
</tr>
<tr>
<td>callStr</td>
<td>The SQL statement used to replicate the training.</td>
</tr>
</tbody>
</table>

**Summarizing Models**

1. Find the model you want to summarize.

```sql
=> SELECT * FROM v_catalog.models WHERE model_name='svm_class';
```

<table>
<thead>
<tr>
<th>model_id</th>
<th>model_name</th>
<th>schema_id</th>
<th>schema_name</th>
<th>owner_id</th>
<th>owner_name</th>
<th>category</th>
<th>model_type</th>
<th>is_complete</th>
<th>create_time</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273715226</td>
<td>svm_class</td>
<td>45035996273704980</td>
<td>public</td>
<td>45035996273704962</td>
<td>dbadmin</td>
<td>VERTICA_MODELS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. View the model summary.

```sql
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='svm_class');
```

<table>
<thead>
<tr>
<th>details</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictor</td>
</tr>
<tr>
<td>Intercept</td>
</tr>
<tr>
<td>cyl</td>
</tr>
<tr>
<td>mpg</td>
</tr>
<tr>
<td>wt</td>
</tr>
<tr>
<td>hp</td>
</tr>
</tbody>
</table>

```sql
=> SELECT svm_classifier('public.svm_class', 'mtcars_train', 'am', 'cyl, mpg, wt, hp, gear' USING PARAMETERS exclude_columns='gear', C=1, max_iterations=100, epsilon=0.001);
```

<table>
<thead>
<tr>
<th>Additional Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>accepted_row_count</td>
</tr>
<tr>
<td>rejected_row_count</td>
</tr>
<tr>
<td>iteration_count</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- **GET_MODEL_SUMMARY**

Viewing Models

Vertica stores the models you create in the `V_CATALOG.MODELS` system table.

You can query `V_CATALOG.MODELS` to view information about the models you have created:

```sql
=> SELECT * FROM V_CATALOG.MODELS;
```

<table>
<thead>
<tr>
<th>model_id</th>
<th>model_name</th>
<th>schema_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273765414</td>
<td>mySvmClassModel</td>
<td>45035996273704978</td>
</tr>
</tbody>
</table>
### See Also

- **MODELS**
Geospatial Analytics

Vertica provides functions that allows you to manipulate complex two- and three-dimensional spatial objects. These functions follow the Open Geospatial Consortium (OGC) standards. Vertica also provides data types and SQL functions that allow you to specify and store spatial objects in a database according to OGC standards.

Convert Well-Known Text (WKT) and Well-Known Binary (WKB)

Convert WKT and WKB.

Optimized Spatial Joins

Perform fast spatial joins using ST_Intersects and STV_Intersects.

Load and Export Spatial Data From Shapefiles

Easily load and export shapefiles.

Store and Retrieve Objects

Determine if:

- An object contains self-intersection or self-tangency points.
- One object is entirely within another object, such as a point within a polygon.

Test the relationships between objects

For example, if they intersect or touch:
- Identify the boundary of an object.
- Identify vertices of an object.

**Calculate**

- Shortest distance between two objects.
- Size of an object (length, area).
- Centroid for one or more objects.
- Buffer around one or more objects.

**Best Practices for Geospatial Analytics**

Vertica recommends the following best practices when performing geospatial analytics in Vertica.

**Performance Optimization**

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the minimum column size for spatial data.</td>
<td>Performance degrades as column widths increase. When creating columns for your spatial data, use the smallest size column that can accommodate your data. For example, use GEOMETRY(85) for point data.</td>
</tr>
<tr>
<td>Use GEOMETRY types where possible.</td>
<td>Performance of functions on GEOGRAPHY types is slower than functions that support GEOMETRY types. Use GEOMETRY types where possible.</td>
</tr>
<tr>
<td>To improve the performance of the following functions, sort projections on spatial columns:</td>
<td>You may improve the pruning efficiency of these functions by sorting the projection on the GEOMETRY column. However, sorting on a large GEOMETRY column may slow down data load.</td>
</tr>
</tbody>
</table>
### Spatial Joins with Points and Polygons

Vertica provides two ways to identify whether a set of points intersect with a set of polygons. Depending on the size of your data set, choose the approach that gives the best performance.

For a detailed example of best practices with spatial joins, see [Best Practices for Spatial Joins](#).

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a spatial index only when performing spatial joins with STV_Intersect.</td>
<td>Spatial indexes should only be used with STV_Intersect. Creating a spatial index and then performing spatial joins with ST_Intersects will not improve performance.</td>
</tr>
<tr>
<td>Use the STV_Intersect function when you intersect a set of points with a set of polygons.</td>
<td>Determine if a set of points intersects with a set of polygons in a medium to large data set. First, create a spatial index using STV_Create_Index. Then, use one of the STV_Intersect functions to return the set of pairs that intersect. Spatial indexes provide the best performance for accessing a large number of polygons.</td>
</tr>
<tr>
<td>When using the STV_Intersect transform function, partition the data and use an OVER(PARTITION BEST) clause.</td>
<td>The STV_Intersect Transform Function does not require that you partition the data. However, you may improve performance by partitioning the data and using an OVER(PARTITION BEST) clause.</td>
</tr>
</tbody>
</table>

### Spatial Indexes

The STV_Create_Index function can consume large amounts of processing time and memory. When you index new data for the first time, monitor memory usage to be sure it stays within
safe limits. Memory usage depends on:

- Number of polygons
- Number of vertices
- Amount of overlap among polygons

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment polygon data when a table contains a large number of polygons.</td>
<td>Segmenting the data allows the index creation process to run in parallel. This is advantageous because sometimes STV_Createt_Index tasks cannot be completed when large tables that are not segmented prior to index creation.</td>
</tr>
</tbody>
</table>
| Adjust STV_Create_Index parameters as needed for memory allocation and CPU usage. | The max_mem_mb parameter can affect the resource usage of STV_Create_Index. max_mem_mb assigns a limit to the amount of memory that STV_Create_Index can allocate. **Default value: 256**
**Valid values:** Any value less than or equal to the amount of memory in the GENERAL resource pool. Assigning a higher value results in an error. |
| Make changes if STV_Create_Index cannot allocate 300 MB memory. | Before STV_Create_Index starts creating the index, it tries to allocate about 300 MB of memory. If that much memory is not available, the function fails. If you get a failure message, try these solutions:  
  - Create the index at a time of less load on the system.  
  - Avoid concurrent index creation.  
  - Add more memory to your system. |
<p>| Create misplaced indexes again, if needed. | When you back up your Vertica database, spatial index files are not included. If you misplace an index, use STV_Create_Index to re-create it. |
| Use STV_Refresh_Index to add new or updated polygons to an existing | Instead of rebuilding your spatial index each time you add new or updated polygons to a table, you |</p>
<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>index.</td>
<td>can use STV_Refresh_Index to append the polygons to your existing spatial index.</td>
</tr>
</tbody>
</table>

### Checking Polygon Validity

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run ST_IsValid to check if polygons are valid.</td>
<td>Many spatial functions do not check the validity of polygons.</td>
</tr>
<tr>
<td>Run ST_IsValid on all polygons to determine if they are valid.</td>
<td></td>
</tr>
<tr>
<td>If your object is not valid, run STV_IsValidReason to get information about the location of the invalid polygon.</td>
<td></td>
</tr>
<tr>
<td>For more information, see <a href="#">Ensuring Polygon Validity Before Creating or Refreshing an Index</a>.</td>
<td></td>
</tr>
</tbody>
</table>

### Spatial Objects

Vertica implements several data types for storing spatial objects, Well-Known Text (WKT) strings, and Well-Known Binary (WKB) representations. These data types include:

- **Supported Spatial Objects**
- **Spatial Reference Identifiers (SRIDs)**

### Supported Spatial Objects

Vertica supports two spatial data types. These data types store two- and three-dimensional spatial objects in a table column:
- **GEOMETRY**: Spatial object with coordinates expressed as \((x, y)\) pairs, defined in the Cartesian plane. All calculations use Cartesian coordinates.

- **GEOGRAPHY**: Spatial object defined as on the surface of a perfect sphere, or a spatial object in the WGS84 coordinate system. Coordinates are expressed in longitude/latitude angular values, measured in degrees. All calculations are in meters. For perfect sphere calculations, the sphere has a radius of 6371 kilometers, which approximates the shape of the earth.

**Note:** Some spatial programs use an ellipsoid to model the earth, resulting in slightly different data.

The maximum size of a GEOMETRY or GEOGRAPHY data type is 10,000,000 bytes (10 MB). You cannot use either data type as a table's primary key.

For details, see Geospatial Analytics in Analyzing Data.

### Spatial Reference Identifiers (SRIDs)

A **spatial reference identifier** (SRID) is an integer value that represents a method for projecting coordinates on the plane. A SRID is metadata that indicates the coordinate system in which a spatial object is defined.

Geospatial functions using Geometry arguments must contain the same SRID. If the functions do not contain the same SRID, then the query returns an error.

For example, in this query the two points have different SRIDs. As a result the query returns an error:

```
=> SELECT ST_Distance(ST_GeomFromText('POINT(34 9)', 2749), ST_GeomFromText('POINT(70 12)', 3359));
ERROR 5861: Error calling processBlock() in User Function ST_Distance at [src/Distance.cpp:65], error code: 0, message: Geometries with different SRIDs found: 2749, 3359
```

### Supported SRIDs

Vertica supports SRIDs derived from the EPSG standards. Geospatial functions using Geometry arguments must use supported SRIDs when performing calculations. SRID values of 0 to \(2^{32}-1\) are valid. Queries with SRID values outside of this range will return an error.
Working with Spatial Objects in Tables

- Defining Table Columns for Spatial Data
- Exporting Spatial Data from a Table
- Identifying Null Spatial Objects
- Loading Spatial Data from Shapefiles
- Loading Spatial Data into Tables Using COPY
- Retrieving Spatial Data from a Table as Well-Known Text (WKT)

Defining Table Columns for Spatial Data

To define columns to contain GEOMETRY and GEOGRAPHY data, use this command:

```
=> CREATE TABLE [[db-name.]schema.]table-name {
    column-name GEOMETRY[(length)],
    column-name GEOGRAPHY[(length)];
```

If you omit the length specification, the default column size is 1 MB. The maximum column size is 10 MB. The upper limit is not enforced, but the geospatial functions can only accept or return spatial data up to 10 MB.

You cannot modify the size or data type of a GEOMETRY or GEOGRAPHY column after creation. If the column size you created is not sufficient, create a new column with the desired size. Then copy the data from the old column, and drop the old column from the table.

You cannot import data to or export data from tables that contain spatial data from another Vertica database.

**Important:** A column width that is too large could impact performance. Use a column width that fits the data without being excessively large. See `STV_MemSize`.

Exporting Spatial Data from a Table

You can export spatial data from a table in your Vertica database to a shapefile.

To export spatial data from a table to a shapefile:
1. As the superuser, set the shapefile export directory.

```sql
=> SELECT STV_SetExportShapefileDirectory(USING PARAMETERS path = '/home/geo/temp');

---

STV_SetExportShapefileDirectory

SUCCESS. Set shapefile export directory: [/home/geo/temp]
(1 row)
```

2. Export your spatial data to a shapefile.

```sql
=> SELECT STV_Export2Shapefile(*
    USING PARAMETERS shapefile = 'visualizations/city-data.shp',
    shape = 'Polygon') OVER() FROM spatial_data;

<table>
<thead>
<tr>
<th>Rows Exported</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>185873</td>
<td>v_geo-db_node0001: /home/geo/temp/visualizations/city-data.shp</td>
</tr>
</tbody>
</table>
(1 row)
```

- The value asterisk (*) is the equivalent to listing all columns in the FROM clause.
- You can specify sub-directories when exporting your shapefile.
- Your shapefile must end with the file extension .shp.

3. Verify that three files now appear in the shapefile export directory.

```bash
$ ls
city-data.dbf  city-data.shp  city-data.shx
```

### Identifying Null Spatial Objects

You can identify null GEOMETRY and GEOGRAPHY objects using the Vertica IS NULL and IS NOT NULL constructs.

This example uses the following table, where the row with id=2 has a null value in the `geog` field.

```sql
=> SELECT id, ST_AsText(geom), ST_AsText(geog) FROM locations
    ORDER BY 1 ASC;

<table>
<thead>
<tr>
<th>id</th>
<th>ST_AsText</th>
<th>ST_AsText</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (2 3)</td>
<td>POINT (-85 15)</td>
</tr>
<tr>
<td>2</td>
<td>POINT (4 5)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>POLYGON ((-1 2, 0 3, 1 2, -1 2))</td>
<td>POLYGON ((-24 12, -15 23, -20 27, -24 12))</td>
</tr>
<tr>
<td>4</td>
<td>LINESTRING (-1 2, 1 5)</td>
<td>LINESTRING (-42.74 23.98, -62.19 23.78)</td>
</tr>
</tbody>
</table>
(4 rows)
```

Identify all the rows that have a null `geog` value:
=> SELECT id, ST_AsText(geom), (ST_AsText(geog) IS NULL) FROM locations
ORDER BY 1 ASC;

| id | ST_AsText | ?column?
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (2 3)</td>
<td>f</td>
</tr>
<tr>
<td>2</td>
<td>POINT (4 5)</td>
<td>t</td>
</tr>
<tr>
<td>3</td>
<td>POLYGON ((-1 2, 0 3, 1 2, -1 2))</td>
<td>f</td>
</tr>
<tr>
<td>4</td>
<td>LINESTRING (-1 2, 1 5)</td>
<td>f</td>
</tr>
</tbody>
</table>

(4 rows)

Identify the rows where the geog value is not null:

=> SELECT id, ST_AsText(geom), (ST_AsText(geog) IS NOT NULL) FROM locations
ORDER BY 1 ASC;

<table>
<thead>
<tr>
<th>id</th>
<th>st_asText</th>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (2 3)</td>
<td>t</td>
</tr>
<tr>
<td>2</td>
<td>POINT (4 5)</td>
<td>f</td>
</tr>
<tr>
<td>3</td>
<td>LINESTRING (-1 2, 1 5)</td>
<td>t</td>
</tr>
<tr>
<td>4</td>
<td>POLYGON ((-1 2, 0 3, 1 2, -1 2))</td>
<td>t</td>
</tr>
</tbody>
</table>

(4 rows)

Loading Spatial Data from Shapefiles

Vertica provides the capability to load and parse spatial data that is stored in shapefiles. Shapefiles describe points, lines, and polygons. A shapefile is made up of three required files; all three files must be present and in the same directory to define the geometries:

- .shp—Contains the geometry data.
- .shx—Contains the positional index of the geometry.
- .dbf—Contains the attributes for each geometry.

To load spatial data from a shapefile:

1. Use STV_ShpCreateTable to generate a CREATE TABLE statement.

=> SELECT STV_ShpCreateTable ( USING PARAMETERS file = '/home/geo/temp/shp-files/spatial_data.shp')
   OVER() AS spatial_data;

CREATE TABLE spatial_data(
  gid IDENTITY(64) PRIMARY KEY,
  uniq_id INT8,
  geom GEOMETRY(85)
);
(5 rows)
2. Create the table.

```sql
CREATE TABLE spatial_data(
    gid IDENTITY(64) PRIMARY KEY,
    uniq_id INT8,
    geom GEOMETRY(85));
```

3. Load the shapefile.

```sql
COPY spatial_data WITH SOURCE STV_ShpSource(file='/home/geo/temp/shp-files/spatial_data.shp')
PARSER STV_ShpParser();
```

### Supported Shapefile Shape Types

The following table lists the shapefile shape types that Vertica supports.

<table>
<thead>
<tr>
<th>Shapefile Shape Type</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null shape</td>
<td>Yes</td>
</tr>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Polyline</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>MultiPoint</td>
<td>Yes</td>
</tr>
<tr>
<td>PointZ</td>
<td>No</td>
</tr>
<tr>
<td>PolylineZ</td>
<td>No</td>
</tr>
<tr>
<td>PolygonZ</td>
<td>No</td>
</tr>
<tr>
<td>MultiPointZ</td>
<td>No</td>
</tr>
<tr>
<td>PointM</td>
<td>No</td>
</tr>
<tr>
<td>PolylineM</td>
<td>No</td>
</tr>
<tr>
<td>PolygonM</td>
<td>No</td>
</tr>
<tr>
<td>MultiPointM</td>
<td>No</td>
</tr>
</tbody>
</table>
Loading Spatial Data into Tables Using COPY

You can load spatial data into a table in Vertica using a COPY statement.

To load data into Vertica using a COPY statement:

1. Create a table.

    ```sql
    => CREATE TABLE spatial_data (id INTEGER, geom GEOMETRY(200));
    CREATE TABLE
    ```

2. Create a text file named `spatial.dat` with the following data.

    ```
    1|POINT(2 3)
    2|LINESTRING(-1 2, 1 5)
    3|POLYGON((-1 2, 0 3, 1 2, -1 2))
    ```

3. Use COPY to load the data into the table.

    ```sql
    => COPY spatial_data (id, gx FILLER LONG VARCHAR(605), geom AS ST_GeomFromText(gx)) FROM LOCAL 'spatial.dat';
    Rows Loaded
    ----------
    3
    (1 row)
    ```

   The statement specifies a LONG VARCHAR(32000000) filler, which is the maximum size of WKT. You must specify a filler value large enough to hold the largest WKT you want to insert into the table.

Retrieving Spatial Data from a Table as Well-Known Text (WKT)

GEOMETRY and GEOGRAPHY data is stored in Vertica tables as LONG VARBINARY, which isn't human readable. You can use `ST_AsText` to return the spatial data as Well-Known Text (WKT).

To return spatial data as WKT:

```sql
=> SELECT id, ST_AsText(geom) AS WKT FROM spatial_data;
    id | WKT
    -- | ----
    1  | POINT(2 3)
    2  | LINESTRING(-1 2, 1 5)
    3  | POLYGON((-1 2, 0 3, 1 2, -1 2))
```
Working with GeoHash Data

Vertica supports **GeoHashes**. A GeoHash is a geocoding system for hierarchically encoding increasingly granular spatial references. Each additional character in a GeoHash drills down to a smaller section of a map.

You can use Vertica to generate spatial data from GeoHashes and GeoHashes from spatial data. Vertica supports the following functions for use with GeoHashes:

- **ST_Geohash** - Returns a GeoHash in the shape of the specified geometry.
- **ST_GeomFromGeoHash** - Returns a polygon in the shape of the specified GeoHash.
- **ST_PointFromGeoHash** - Returns the center point of the specified GeoHash.

For example, to generate a full precision and partial precision GeoHash from a single point.

```sql
=> SELECT ST_Geohash(ST_GeographyFromText('POINT(3.14 -1.34)'), LENGTH(ST_Geohash(ST_GeographyFromText('POINT(3.14 -1.34)'))),
ST_Geohash(ST_GeographyFromText('POINT(3.14 -1.34)')) USING PARAMETERS
numchars=5) partial_hash;

<table>
<thead>
<tr>
<th>partial_hash</th>
<th>LENGTH</th>
<th>partial_hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>kpfo3mz5sks75010</td>
<td>20</td>
<td>kpfo3mz5sks75010</td>
</tr>
</tbody>
</table>
```

This example shows how to generate a GeoHash from a multipoint point object. The returned polygon is a geometry object of the smallest tile that encloses that GeoHash.

```sql
=> SELECT ST_AsText(ST_GeomFromGeoHash(ST_Geohash(ST_GeomFromText('MULTIPOINT(0 0, 0.0001 0.0001)')))) AS region_1,
ST_AsText(ST_GeomFromGeoHash(ST_Geohash(ST_GeomFromText('MULTIPOINT(0.0001 0.0003 0.0002)')))) AS region_2;

--

region_1 | POLYGON ((0 0, 0.0137329101562 0, 0.00137329101562 0.00137329101562, 0 0.0137329101562, 0 0))
region_2 | POLYGON ((0 0, 0.010986328125 0, 0.010986328125 0.0054931640625, 0 0.0054931640625, 0 0))
```
Spatial Joins with ST_Intersects and STV_Intersect

Spatial joins allow you to identify spatial relationships between two sets of spatial data. For example, you can use spatial joins to:

- Calculate the density of mobile calls in various regions to determine the location of a new cell phone tower.
- Identify homes that fall within the impact zone of a hurricane.
- Calculate the number of users who live within a certain ZIP code.
- Calculate the number of customers in a retail store at any given time.

Best Practices for Spatial Joins

Use these best practices to improve overall performance and optimize your spatial queries.

Best practices for using spatial joins in Vertica include:

- Table segmentation to speed up index creation
- Adequately sizing a geometry column to store point data
- Loading Well-Known Text (WKT) directly into a Geometry column, using STV_GeometryPoint in a COPY statement
- Using OVER (PARTITION BEST) with STV_Intersect transform queries

Best Practices Example

Note: The following example was originally published in a Vertica blog post about using spatial data in museums. To read the entire blog, go to:


Before performing the steps in the following example, download place_output.csv.zip from the Vertica Place GitHub repository (https://github.com/vertica/Vertica-Geospatial). You need to use the data set from this repository.
1. Create the table for the polygons. Use a GEOMETRY column width that fits your data without being excessively large. A good column-width fit improves performance. In addition, segmenting the table by HASH provides the advantages of parallel computation.

```sql
CREATE TABLE artworks (gid int, g GEOMETRY(700)) SEGMENTED BY HASH(gid) ALL NODES;
```

2. Use a copy statement with ST_Buffer to create and load the polygons on which to run the intersect. By using ST_Buffer in your copy statement, you can use that function to create the polygons.

```sql
COPY artworks(gid, gx FILLER LONG VARCHAR, g AS ST_Buffer(ST_GeomFromText(gx),8)) FROM STDIN
DELIMITER ','.;
```

3. Create a table for the location data, represented by points. You can store point data in a GEOMETRY column of 100 bytes. Avoid over-fitting your GEOMETRY column. Doing so can significantly degrade spatial intersection performance. Also, segment this table by HASH, to take advantage of parallel computation.

```sql
CREATE TABLE usr_data (gid identity, usr_id int, date_time timestamp, g GEOMETRY(100)) SEGMENTED BY HASH(gid) ALL NODES;
```

4. During the copy statement, transform the raw location data to GEOMETRY data. You must perform this transformation because your location data needs to use the GEOMETRY data type. Use the function STV_GeometryPoint to transform the x and y columns of the source table.

```sql
COPY usr_data (usr_id, date_time, x FILLER LONG VARCHAR, y FILLER LONG VARCHAR, g AS STV_GeometryPoint(x, y)) FROM LOCAL 'place_output.csv' DELIMITER ',' ENCLOSED BY '';
```

5. Create the spatial index for the polygons. This index helps you speed up intersection calculations.

```sql
SELECT STV_Create_Index(gid, g USING PARAMETERS index='art_index', overwrite=true) OVER() FROM artworks;
```

6. Write an analytic query that returns the number of intersections per polygon. Specify that Vertica ignore any usr_id that intersects less than 20 times with a given polygon.
```sql
> SELECT pol_gid,
>     COUNT(DISTINCT(usr_id)) AS count_user_visit
> FROM
>     (SELECT pol_gid,
>         usr_id,
>         COUNT(usr_id) AS user_points_in
>     FROM
>         (SELECT STV_Intersect(usr_id, g USING PARAMETERS INDEX='art_index') OVER(PARTITION BEST) AS pol_gid
>         FROM usr_data
>             WHERE date_time BETWEEN '2014-07-02 09:30:20' AND '2014-07-02 17:05:00') AS c
>     GROUP BY pol_gid
>     ORDER BY count_user_visit DESC;
```

**Optimizations in the Example Query**

This query has the following optimizations:

- The time predicated appears in the subquery.
- Using the location data table avoids the need for an expensive join.
- The query uses OVER (PARTITION BEST), to improve performance by partitioning the data.
- The user_points_in provides an estimate of the combined time spent intersecting with the artwork by all visitors.

**Ensuring Polygon Validity Before Creating or Refreshing an Index**

When Vertica creates or updates a spatial index it does not check polygon validity. To prevent getting invalid results when you query your spatial index, you should check the validity of your polygons prior to creating or updating your spatial index.

The following example shows you how to check the validity of polygons.

1. Create a table and load spatial data.

```sql
> CREATE TABLE polygon_validity_test (gid INT, geom GEOMETRY);
CREATE TABLE
> COPY polygon_validity_test (gid, gx FILLER LONG VARCHAR, geom AS St_GeomFromText(gx)) FROM STDIN;
```
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.

```
>> 2 | POLYGON((-31 74,8 70,8 58,-36 53,-31 74))
>> 3 | POLYGON((-38 58,4 13,11 45,0 65,-38 50))
>> 4 | POLYGON((-12 42,-12 42,27 48,14 26,-12 42))
>> 5 | POLYGON((0 0,1 0,2 1,1 1,0 0))
>> 6 | POLYGON((3 3,2 2,1 1,2 3,3 3))
```

2. Use ST_IsValid and STV_IsValidReason to find any invalid polygons.

```
=> SELECT gid, ST_IsValid(geom), STV_IsValidReason(geom) FROM polygon_validity_test;

<table>
<thead>
<tr>
<th>gid</th>
<th>ST_IsValid</th>
<th>STV_IsValidReason</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>f</td>
<td>Self-intersection at or near POINT (2 1)</td>
</tr>
<tr>
<td>2</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>f</td>
<td>Self-intersection at or near POINT (0 0)</td>
</tr>
</tbody>
</table>

(5 rows)
```

Now that we have identified the invalid polygons in our table, there are a couple different ways you can handle the invalid polygons when creating or refreshing a spatial index.

Filtering Invalid Polygons Using a WHERE Clause

This method is slower than filtering before creating an index because it checks the validity of each polygon at execution time.

The following example shows you how to exclude invalid polygons using a WHERE clause.

```
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index = 'valid_polygons') OVER() 
   FROM polygon_validity_test 
   WHERE ST_IsValid(geom) = 't';
```

Filtering Invalid Polygons Before Creating or Refreshing an Index

This method is faster than filtering using a WHERE clause because you incur the performance cost prior to building the index.

The following example shows you how to exclude invalid polygons by creating a new table excluding invalid polygons.

```
=> CREATE TABLE polygon_validity_clean AS 
   SELECT * 
   FROM polygon_validity_test 
   WHERE ST_IsValid(geom) = 't';
```
CREATE TABLE
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index = 'valid_polygons') OVER()
    FROM polygon_validity_clean;

**STV_Intersect: Scalar Function vs. Transform Function**

The `STV_Intersect` functions are similar in purpose, but you use them differently.

<table>
<thead>
<tr>
<th><strong>STV_Intersect Function Type</strong></th>
<th><strong>Description</strong></th>
<th><strong>Performance</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>Matches a point to a polygon. If several polygons contain the point, this function returns a gid value. The result is a polygon gid or, if no polygon contains the point, the result is NULL.</td>
<td>Eliminates points that do not intersect with any indexed polygons, avoiding unnecessary comparisons.</td>
</tr>
<tr>
<td>Transform</td>
<td>Matches a point to all the polygons that contain it. When a point does not intersect with any polygon in the index, the function returns no rows.</td>
<td>Processes all input points regardless of whether or not they intersect with the indexed polygons.</td>
</tr>
</tbody>
</table>

In the following example, the `STV_Intersect` scalar function compares the points in the `points` table to the polygons in a spatial index named `my_polygons`. `STV_Intersect` returns all points and polygons that match exactly:

```sql
=> SELECT gid AS pt_gid,
    STV_Intersect(geom USING PARAMETERS index='my_polygons') AS pol_gid
FROM points ORDER BY pt_gid;
```

<table>
<thead>
<tr>
<th>pt_gid</th>
<th>pol_gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>2</td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>3</td>
</tr>
<tr>
<td>106</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
</tr>
</tbody>
</table>

The following example shows how to use the `STV_Intersect` transform function to return information about the three point-polygon pairs that match and each of the polygons they match:
```sql
=> SELECT STV_Intersect(gid, geom
    USING PARAMETERS index='my_polygons')
   OVER (PARTITION BEST) AS (pt_gid, pol_id)
FROM points;
  pt_gid | pol_id
---------+--------
     100  |    1
     100  |    2
     100  |    3
     102  |    2
     105  |    3
(3 rows)
```

See Also

- STV_Intersect Scalar Function
- STV_Intersect Transform Function

Performing Spatial Joins with STV_Intersect Functions

Suppose you want to process a medium-to-large spatial data set and determine which points intersect with which polygons. In that case, first create a spatial index using STV_Create_Index. A spatial index provides efficient access to the set of polygons.

Then, use the STV_Intersect scalar or transform function to identify which point-polygon pairs match.

Spatial Indexes and STV_Intersect

Before performing a spatial join using one of the STV_Intersect functions, you must first run STV_Create_Index to create a database object that contains information about polygons. This object is called a spatial index of the set of polygons. The spatial index improves the time it takes for the STV_Intersect functions to access the polygon data.

Vertica creates spatial indexes in a global space. Thus, any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Vertica provides functions that work with spatial indexes:

- STV_Create_Index—Stores information about polygons in an index to improve performance.
- STV_Describe_Index—Retrieves information about an index.
- STV_Drop_Index—Deletes a spatial index.
- **STV_Refresh_Index**—Refreshes a spatial index.
- **STV_Rename_Index**—Renames a spatial index.

**When to Use ST_Intersects vs. STV_Intersect**

Vertica provides two capabilities to identify whether a set of points intersect with a set of polygons. Depending on the size of your data set, choose the approach that gives the best performance:

- When comparing a set of geometries to a single geometry to see if they intersect, use the **ST_Intersects** function.

- To determine if a set of points intersects with a set of polygons in a medium-to-large data set, first create a spatial index using **STV_Create_Index**. Then, use one of the **STV_Intersect** functions to return the set of pairs that intersect.

**Note:** You can only perform spatial joins on GEOMETRY data.

**Performing Spatial Joins with ST_Intersects**

The **ST_Intersects** function determines if two GEOMETRY objects intersect or touch at a single point.

Use **ST_Intersects** when you want to identify if a small set of geometries in a column intersect with a given geometry.
**Example**

The following example uses ST_Intersects to compare a column of point geometries to a single polygon. The table that contains the points has 1 million rows.

ST_Intersects returns only the points that intersect with the polygon. Those points represent about 0.01% of the points in the table:

```sql
=> CREATE TABLE points_1m(gid IDENTITY, g GEOMETRY(100)) ORDER BY g;
=> COPY points_1m(wkt FILLER LONG VARCHAR(100), g AS ST_GeomFromText(wkt))
    FROM LOCAL '/data/points.dat' DIRECT;
Rows Loaded
------------------
1000000
(1 row)
=> SELECT ST_AsText(g) FROM points_1m WHERE
  ST_Intersects
  (g,
   ST_GeomFromText('POLYGON((-71 42, -70.9 42, -70.9 42.1, -71 42.1, -71 42))'))

st_astext
-------------------
POINT (-70.97532 42.03538)
POINT (-70.97421 42.0376)
POINT (-70.99004 42.07538)
POINT (-70.99477 42.08454)
POINT (-70.99088 42.08177)
POINT (-70.98643 42.07593)
POINT (-70.98832 42.07982)
POINT (-70.95921 42.00982)
POINT (-70.95115 42.02177)
...
(116 rows)
```

OpenText recommends that you test the intersections of two columns of geometries by creating a spatial index. Use one of the STV_Intersect functions as described in STV_Intersect: Scalar Function vs. Transform Function.

**Working with Spatial Objects from Client Applications**

The Vertica client driver libraries provide interfaces for connecting your client applications to your Vertica database. The drivers simplify exchanging data for loading, report generation, and other common database tasks.

There are three separate client drivers:
Vertica Place supports the following new data types:

- LONG VARCHAR
- LONG VARBINARY
- GEOMETRY
- GEOGRAPHY

The client driver libraries support these data types; the following sections describe that support and provide examples.

**Using LONG VARCHAR and LONG VARBINARY Data Types with ODBC**

The ODBC drivers support the LONG VARCHAR and LONG VARBINARY data types similarly to VARCHAR and VARBINARY data types. When binding input or output parameters to a LONG VARCHAR or LONG VARBINARY column in a query, use the SQL_LONGVARCHAR and SQL_LONGVARBINARY constants to set the column's data type. For example, to bind an input parameter to a LONG VARCHAR column, you would use a statement that looks like this:

```c
rc = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR, 80000, 0, (SQLPOINTER)myLongString, sizeof(myLongString), NULL);
```

**Note:** Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).
Using LONG VARCHAR and LONG VARBINARY Data Types with JDBC

Using LONG VARCHAR and LONG VARBINARY data types in a JDBC client application is similar to using VARCHAR and VARBINARY data types. The JDBC driver transparently handles the conversion (for example, between a Java String object and a LONG VARCHAR). The following example code demonstrates inserting and retrieving a LONG VARCHAR string. It uses the JDBC Types class to determine the data type of the string returned by Vertica, although it does not actually need to know whether the database column is a LONG VARCHAR or just a VARCHAR in order to retrieve the value.

```java
import java.sql.*;
import java.util.Properties;

public class LongVarcharExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");

        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            // establish connection and make a table for the data.
            Statement stmt = conn.createStatement();

            // How long we want the example string to be. This is
            // larger than can fit into a traditional VARCHAR (which is limited
            // to 65000.
            int length = 100000;

            // Create a table with a LONG VARCHAR column that can store
            // the string we want to insert.
            stmt.execute("DROP TABLE IF EXISTS longtable CASCADE");
            stmt.execute("CREATE TABLE longtable (text LONG VARCHAR(" + length + "))");

            // Build a long string by appending an integer to a string builder
            // until we hit the size limit. Will result in a string
            // containing 01234567890123....
            StringBuilder sb = new StringBuilder(length);
            for (int i = 0; i < length; i++) {
                sb.append(i % 10);
            }
```
String value = sb.toString();

System.out.println("String value is " + value.length() + " characters long.");

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO longtable (text) VALUES (?)");

try {
    // Insert LONG VARCHAR value
    System.out.println("Inserting LONG VARCHAR value");
    pstmt.setString(1, value);
    pstmt.addBatch();
    pstmt.executeBatch();

    // Query the table we created to get the value back.
    ResultSet rs = null;
    rs = stmt.executeQuery("SELECT * FROM longtable");

    // Get metadata about the result set.
    ResultSetMetaData rsmd = rs.getMetaData();
    // Print the type of the first column. Should be
    // LONG VARCHAR. Also check it against the Types class, to
    // recognize it programmatically.
    System.out.println("Column #1 data type is: " +
                        rsmd.getColumnTypeName(1));
    if (rsmd.getColumnType(1) == Types.LONGVARCHAR) {
        System.out.println("It is a LONG VARCHAR");
    } else {
        System.out.println("It is NOT a LONG VARCHAR");
    }

    // Print out the string length of the returned value.
    while (rs.next()) {
        // Use the same getString method to get the value that you
        // use to get the value of a VARCHAR.
        System.out.println("Returned string length: " +
                            rs.getString(1).length());
    }
} catch (SQLException e) {
    System.out.println("Error message: " + e.getMessage());
    return; // Exit if there was an error
}

// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG
VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the
decoded value is less than 32 MB in size. For example, Vertica returns an error if you
attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Using GEOMETRY and GEOGRAPHY Data Types in ODBC

Vertica GEOMETRY and GEOGRAPHY data types are backed by LONG VARBINARY native types and ODBC client applications treat them as binary data. However, these data types have a format that is unique to Vertica. To manipulate this data in your C++ application, you must use the functions in Vertica that convert them to a recognized format.

To convert a WKT or WKB to the GEOMETRY or GEOGRAPHY format, use one of the following SQL functions:

- **ST_GeographyFromText** — Converts a WKT to a GEOGRAPHY type.
- **ST_GeographyFromWKB** — Converts a WKB to a GEOGRAPHY type.
- **ST_GeomFromText** — Converts a WKT to a GEOMETRY type.
- **ST_GeomFromWKB** — Converts a WKB to GEOMETRY type.

To convert a GEOMETRY or GEOGRAPHY object to its corresponding WKT or WKB, use one of the following SQL functions:

- **ST_AsText** — Converts a GEOMETRY or GEOGRAPHY object to a WKT, returns a LONGVARCHAR.
- **ST_AsBinary** — Converts a GEOMETRY or GEOGRAPHY object to a WKB, returns a LONGVARBINARY.

The following code example converts WKT data into GEOMETRY data using ST_GeomFromText and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using ST_AsText and ST_AsBinary.

```cpp
// Compile on Linux using:
// g++ -g -I/opt/vertica/include -L/opt/vertica/lib64 -lodbc -o SpatialData SpatialData.cpp
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <sstream>
// Only needed for Windows clients
```
// include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

// Helper function to print SQL error messages.
template <typename HandleT>
void reportError(int handleTypeEnum, HandleT hdl)
{
  // Get the status records.
  SQLSMALLINT  i, MsgLen;
  SQLRETURN ret2;
  SQLCHAR SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
  SQLINTEGER NativeError;
  i = 1;
  printf("\n");
  while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError, Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {
    printf("error record %d\n", i);
    printf("sqlstate: %s\n", SqlState);
    printf("detailed msg: %s\n", Msg);
    printf("native error code: %d\n", NativeError);
    i++;
  }
  exit(EXIT_FAILURE); // bad form... but Ok for this demo
}

int main()
{
  // Set up the ODBC environment
  SQLRETURN ret;
  SQLENV hd1Env;
  ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
  assert(SQL_SUCCEEDED(ret));
  // Tell ODBC that the application uses ODBC 3.
  ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
  assert(SQL_SUCCEEDED(ret));
  // Allocate a database handle.
  SQLDBC hd1Dbc;
  ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
  assert(SQL_SUCCEEDED(ret));
  // Connect to the database
  printf("Connecting to database.\n");
  const char *dsnName = "ExampleDB";
  const char* userID = "dbadmin";
  const char* passwd = "password123";
  ret = SQLConnect(hd1Dbc, (SQLCHAR*)dsnName, SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
                   (SQLCHAR*)passwd, SQL_NTS);
  if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    reportError<SQLDBC>(SQL_HANDLE_DBC, hd1Dbc);
  } else {
    printf("Connected to database.\n");
  }

  // Disable AUTOCOMMIT
  ret = SQLSetConnectAttr(hd1Dbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
}
// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Drop any previously defined table.
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS polygons", SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

// Run query to create a table to hold a geometry.
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE polygons (id INTEGER PRIMARY KEY, poly GEOMETRY);", SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

// Create the prepared statement. This will insert data into the
// table we created above. It uses the ST_GeomFromText function to convert the
// string-formatted polygon definition to a GEOMETRY datat type.
printf("Creating prepared statement\n");
ret = SQLPrepare(hdlStmt, (SQLTCHAR*)"INSERT INTO polygons (id, poly) VALUES(?, ST_GeomFromText(?))", SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

SQLINTEGER id = 0;
int numBatches = 5;
int rowsPerBatch = 10;

// Polygon definition as a string.
char polygon[] = "polygon((1 1, 1 2, 2 2, 2 1, 1 1));";
// Bind variables to the parameters in the prepared SQL statement
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &id, 0, NULL);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
// Bind polygon string to the geometry column
SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR, strlen(polygon), 0, (SQLPOINTER)polygon, strlen(polygon), NULL);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

// Execute the insert
ret = SQLExecute(hdlStmt);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);
} else {
    printf("Executed batch.\n");
}

// Commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);
} else {
    printf("Committed transaction\n");
}

// Now, create a query to retrieve the geometry.
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
printf("Getting data from table.\n");
// Execute a query to get the id, raw geometry data, and
```c
// the geometry data as a string. Uses the ST_AsText SQL function to
// format raw data back into a string polygon definition
ret = SQLExecDirect(hdlStmt,
    (SQLCHAR*)"select id,ST_AsBinary(poly),ST_AsText(poly) from polygons ORDER BY id;",
    SQL_NTS);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}

SQLINTEGER idval;
// 10MB buffer to hold the raw data from the geometry (10Mb is the maximum
// length of a GEOMETRY)
SQLCHAR* polygonval = (SQLCHAR*)malloc(10485760);
SQLLEN polygonlen, polygonstrlen;
// Buffer to hold a LONGVARCHAR that can result from converting the
// geometry to a string.
SQLTCHAR* polygonstr = (SQLTCHAR*)malloc(33554432);

// Get the results of the query and print each row.
do {
    ret = SQLFetch(hdlStmt);
    if (SQL_SUCCEEDED(ret)) {
        // ID column
        ret = SQLGetData(hdlStmt, 1, SQL_C_LONG, &idval, 0, NULL);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("id: %d\n", idval);
        // The WKB format geometry data
        ret = SQLGetData(hdlStmt, 2, SQL_C_BINARY, polygonval, 10485760,
            &polygonlen);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("Polygon in WKB format: \n");
        // Print each byte of polygonval buffer in hex format.
        for (int z = 0; z < polygonlen; z++)
            printf("%02x ", polygonval[z]);
        printf("\n");
        // Geometry data formatted as a string.
        ret = SQLGetData(hdlStmt, 3, SQL_C_TCHAR, polygonstr, 33554432, &polygonstrlen);
        if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
        printf("Polygon in WKT format: %\n", polygonstr);
    }
} while(SQL_SUCCEEDED(ret));

free(polygonval);
free(polygonstr);
// Clean up
printf("Free handles.\n");
ret = SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
ret = SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
ret = SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
if (!SQL_SUCCEEDED(ret)) {reportError<SQLHDBC>(SQL_HANDLE_STMT, hdlStmt);}
exit(EXIT_SUCCESS);
}
```

The output of running the above example is:

```
Connecting to database.
Connected to database.
```
Creating prepared statement
Executed batch.
Committing transaction
Committed transaction
Getting data from table.
id: 0
Polygon in WKB format: 01 03 00 00 01 00 00 05 00 00 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 f0 3f
Polygon in WKT format: POLYGON ((1 1, 1 2, 2 2, 2 1, 1 1))
Free handles.

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32 MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32 MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

Using GEOMETRY and GEOGRAPHY Data Types in JDBC

Vertica GEOMETRY and GEOGRAPHY data types are backed by LONG VARBINARY native types and JDBC client applications treat them as binary data. However, these data types have a format that is unique to Vertica. To manipulate this data in your Java application, you must use the functions in Vertica that convert them to a recognized format.

To convert a WKT or WKB to the GEOMETRY or GEOGRAPHY format, use one of the following SQL functions:

- **ST_GeographyFromText**—Converts a WKT to a GEOGRAPHY type.
- **ST_GeographyFromWKB**—Converts a WKB to a GEOGRAPHY type.
- **ST_GeomFromText**—Converts a WKT to a GEOMETRY type.
- **ST_GeomFromWKB**—Converts a WKB to GEOMETRY type.

To convert a GEOMETRY or GEOGRAPHY object to its corresponding WKT or WKB, use one of the following SQL functions:

- **ST_AsText**—Converts a GEOMETRY or GEOGRAPHY object to a WKT, returns a LONGVARCHAR.
- **ST_AsBinary**—Converts a GEOMETRY or GEOGRAPHY object to a WKB, returns a LONGVARBINARY.
The following code example converts WKT and WKB data into GEOMETRY data using `ST_GeomFromText` and `ST_GeomFromWKB` and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using `ST_AsText` and `ST_AsBinary`.

```java
import java.io.InputStream;
import java.io.Reader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
public class GeospatialDemo
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("com.vertica.jdbc.Driver");
        Connection conn =
            DriverManager.getConnection("jdbc:vertica://localhost:5433/db",
            "user", "password");
        conn.setAutoCommit(false);
        Statement stmt = conn.createStatement();
        stmt.execute("CREATE TABLE polygons(id INTEGER PRIMARY KEY, poly GEOMETRY) ");

        int id = 0;
        int numBatches = 5;
        int rowsPerBatch = 10;

        //batch inserting WKT data
        PreparedStatement pstmt = conn.prepareStatement("INSERT INTO polygons(id, poly) VALUES (?, ST_GeomFromText(?))");
        for(int i = 0; i < numBatches; i++)
        {
            for(int j = 0; j < rowsPerBatch; j++)
            {
                //Insert your own WKT data here
                pstmt.setInt(1, id++);
                pstmt.setString(2, "polygon((1 1, 1 2, 2 2, 2 1, 1 1))");
                pstmt.addBatch();
            }
            pstmt.executeBatch();
        }
        conn.commit();
        pstmt.close();

        //batch insert WKB data
        pstmt = conn.prepareStatement("INSERT INTO polygons(id, poly) VALUES (?, ST_GeomFromWKB(?))");
        for(int i = 0; i < numBatches; i++)
        {
            for(int j = 0; j < rowsPerBatch; j++)
            {
                //Insert your own WKB data here
                byte[] wkb = getWKB();
                pstmt.setInt(1, id++);
                pstmt.setBytes(2, wkb);
            }
        }
    }
}
```
```
    pstmt.addBatch();
    }
    pstmt.executeBatch();
    }
    conn.commit();
    pstmt.close();
    //selecting data as WKT
    ResultSet rs = stmt.executeQuery("select ST_AsText(poly) from polygons");
    while(rs.next())
    {
        String wkt = rs.getString(1);
        Reader wktReader = rs.getCharacterStream(1);
        //process the wkt as necessary
    }
    rs.close();
    //selecting data as WKB
    rs = stmt.executeQuery("select ST_AsBinary(poly) from polygons");
    while(rs.next())
    {
        byte[] wkb = rs.getBytes(1);
        InputStream wkbStream = rs.getBinaryStream(1);
        //process the wkb as necessary
    }
    rs.close();
    //binding parameters in predicates
    pstmt = conn.prepareStatement("SELECT id FROM polygons WHERE
    ST_Contains(ST_GeomFromText(?), poly)";
    pstmt.setString(1, "polygon((1 1, 1 2, 2 2, 2 1, 1 1))");
    rs = pstmt.executeQuery();
    while(rs.next())
    {
        int pk = rs.getInt(1);
        //process the results as necessary
    }
    rs.close();
    }
    conn.close();
    }
```

Using GEOMETRY and GEOGRAPHY Data Types in ADO.NET

Vertica GEOMETRY and GEOGRAPHY data types are backed by LONG VARBINARY native types and ADO.NET client applications treat them as binary data. However, these data types have a format that is unique to Vertica. To manipulate this data in your C# application, you must use the functions in Vertica that convert them to a recognized format.

To convert a WKT or WKB to the GEOMETRY or GEOGRAPHY format, use one of the following SQL functions:
- **ST_GeographyFromText**—Converts a WKT to a GEOGRAPHY type.
- **ST_GeographyFromWKB**—Converts a WKB to a GEOGRAPHY type.
- **ST_GeomFromText**—Converts a WKT to a GEOMETRY type.
- **ST_GeomFromWKB**—Converts a WKB to GEOMETRY type.

To convert a GEOMETRY or GEOGRAPHY object to its corresponding WKT or WKB, use one of the following SQL functions:

- **ST_AsText**—Converts a GEOMETRY or GEOGRAPHY object to a WKT, returns a LONGVARCHAR.
- **ST_AsBinary**—Converts a GEOMETRY or GEOGRAPHY object to a WKB, returns a LONGVARBINARY.

The following C# code example converts WKT data into GEOMETRY data using ST_GeomFromText and stores it in a table. Later, this example retrieves the GEOMETRY data from the table and converts it to WKT and WKB format using ST_AsText and ST_AsBinary.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "VerticaHost";
            builder.Database = "VMart";
            builder.User = "ExampleUser";
            builder.Password = "password123";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "DROP TABLE IF EXISTS polygons";
            command.ExecuteNonQuery();
            command.CommandText = "CREATE TABLE polygons (id INTEGER PRIMARY KEY, poly GEOMETRY)";
            command.ExecuteNonQuery();
            // Prepare to insert a polygon using a prepared statement. Use the
            // ST_GeomFromText SQL function to convert from WKT to GEOMETRY.
            VerticaTransaction txn = _conn.BeginTransaction();
        }
    }
}
```
command.CommandText = 
"INSERT into polygons VALUES(@id, ST_GeomFromText(@polygon))";
command.Parameters.Add(new VerticaParameter("id", VerticaType.BigInt));
command.Parameters.Add(new VerticaParameter("polygon", VerticaType.VarChar));
command.Prepare();
// Set the values for the parameters
command.Parameters["id"].Value = 0;
// command.Parameters["polygon"].Value = 
// "polygon((1 1, 1 2, 2 2, 2 1, 1 1))";
// Execute the query to insert the value
command.ExecuteNonQuery();

// Now query the table
VerticaCommand query = _conn.CreateCommand();
query.CommandText = 
"SELECT id, ST_AsText(poly), ST_AsBinary(poly) FROM polygons;";
VerticaDataReader dr = query.ExecuteReader();
while (dr.Read())
{
    Console.WriteLine("ID: "+ dr[0]);
    Console.WriteLine("Polygon WKT format data type: "
    + dr.GetDataTypeName(1) + 
    " Value: " + dr[1]);
    // Get the WKB format of the polygon and print it out as hex.
    Console.WriteLine("Polygon WKB format data type: "
    + dr.GetDataTypeName(2));
    Console.WriteLine(" Value: 
    + BitConverter.ToString((byte[])dr[2]));
}
_conn.Close();
}
Spatial Classes

Vertica supports several classes of objects, as defined in the OGC standards.

Point

A location in two-dimensional space that is identified by one of the following:

- X and Y coordinates
- Longitude and latitude values

A point has dimension 0 and no boundary.

Examples

The following example uses a GEOMETRY point:

```sql
=> CREATE TABLE point_geo (gid int, geom GEOMETRY(100));
CREATE TABLE
=> COPY point_geo(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter ',,';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1, POINT(3 5)
>>\.
=> SELECT gid, ST_AsText(geom) FROM point_geo;
gid |      ST_AsText
-----+------------------
   1 |  POINT (3 5)
(1 row)
```

The following example uses a GEOGRAPHY point:

```sql
=> CREATE TABLE point_geog (gid int, geog geography(100));
CREATE TABLE
=> COPY point_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter ',,';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1, POINT(42 71)
>>\.
=> SELECT gid, ST_AsText(geog) FROM point_geog;
gid |      ST_AsText
-----+------------------
   1 |  POINT (42 71)
(1 row)
```
Multipoint

A set of one or more points. A multipoint object has dimension 0 and no boundary.

Examples

The following example uses a GEOMETRY multipoint:

```sql
CREATE TABLE mpoint_geo (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY mpoint_geo(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '\';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
CREATE TABLE
=> SELECT gid, ST_AsText(geom) FROM mpoint_geo;
gid | st_astext
1 | MULTIPoint (7 8, 6 9)
(1 row)
```

The following example uses a GEOGRAPHY multipoint:

```sql
CREATE TABLE mpoint_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY mpoint_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '\';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
CREATE TABLE
=> SELECT gid, ST_AsText(geom) FROM mpoint_geog;
gid | st_astext
1 | MULTIPoint (42 71, 41.4 70)
(1 row)
```

Linestring

One or more connected lines, identified by pairs of consecutive points. A linestring has dimension 1. The boundary of a linestring is a multipoint object containing its start and end points.

The following are examples of linestrings:
Examples

The following example uses the GEOMETRY type to create a table, use copy to load a linestring to the table, and then queries the table to view the linestring:

```sql
=> CREATE TABLE linestring_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY linestring_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|LINESTRING(0 0, 1 1, 2 2, 3 4, 2 4, 1 5)
>>\.
=> SELECT gid, ST_AsText(geom) FROM linestring_geom;
gid | ST_AsText
------------------------
 1 | LINESTRING (0 0, 1 1, 2 2, 3 4, 2 4, 1 5)
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a linestring to the table, and then queries the table to view the linestring:

```sql
=> CREATE TABLE linestring_geog (gid int, geom GEOGRAPHY(1000));
CREATE TABLE
=> COPY linestring_geog(gid, gx filler LONG VARCHAR, geom AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|LINESTRING(42.1 71, 41.4 70, 41.3 72.9, 42.99 71.46, 44.47 73.21)
>>\.
=> SELECT gid, ST_AsText(geom) FROM linestring_geog;
gid | ST_AsText
------------------------
 1 | LINESTRING (42.1 71, 41.4 70, 41.3 72.9, 42.99 71.46, 44.47 73.21)
(1 row)
```
Multilinestring

A collection of zero or more linestrings. A multilinestring has no dimension. The boundary of a multilinestring is a multipoint object containing the start and end points of all the linestrings.

The following are examples of multilinestrings:

![Examples of Multilinestrings](image)

Examples

The following example uses the GEOMETRY type to create a table, use copy to load a multilinestring to the table, and then queries the table to view the multilinestring:

```
=> CREATE TABLE multilinestring_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multilinestring_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|MULTILINESTRING((1 5, 2 4, 5 3, 6 6),(3 5, 3 7))
>>\.
=> SELECT gid, ST_AsText(geom) FROM multilinestring_geom;
gid | ST_AsText
-------------------------------------
1 | MULTILINESTRING ((1 5, 2 4, 5 3, 6 6), (3 5, 3 7))
```
The following example uses the GEOGRAPHY type to create a table, use `copy` to load a multilinestring to the table, and then queries the table to view the multilinestring:

```sql
=> CREATE TABLE multilinestring_geog (gid int, geog GEOGRAPHY(1000));
=> CREATE TABLE
=> COPY multilinestring_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|
End with a backslash and a period on a line by itself.
>>1|MULTILINESTRING((42.1 71, 41.4 70, 41.3 72.9), (42.99 71.46, 44.47 73.21))
>>\
=> SELECT gid, ST_AsText(geog) FROM multilinestring_geog;
+-----+
| gid | ST_AsText                           |
+-----+
| 1    | MULTILINESTRING((42.1 71, 41.4 70, 41.3 72.9), (42.99 71.46, 44.47 73.21)) |
+-----+
(1 row)
```

**Polygon**

An object identified by a set of closed linestrings. A polygon can have one or more holes, as defined by interior boundaries, but all points must be connected. Two examples of polygons are:

```

Inclusive and Exclusive Polygons

Polygons that include their points in clockwise order include all space inside the perimeter of the polygon and exclude all space outside that perimeter. Polygons that include their points in counterclockwise order exclude all space inside the perimeter and include all space outside that perimeter.
```
Examples

The following example uses the GEOMETRY type to create a table, use copy to load a polygon into the table, and then queries the table to view the polygon:

```sql
=> CREATE TABLE polygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY polygon_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON(( 2 6, 2 9, 6 9, 7 7, 4 6, 2 6))
>>>.
=> SELECT gid, ST_AsText(geom) FROM polygon_geom;
gid | ST_AsText
------------------------
 1 | POLYGON((2 6, 2 9, 6 9, 7 7, 4 6, 2 6))
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a polygon into the table, and then queries the table to view the polygon:

```sql
=> CREATE TABLE polygon_geog (gid int, geog GEOGRAPHY(1000));
CREATE TABLE
=> COPY polygon_geog(gid, gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71))
>>>.
=> SELECT gid, ST_AsText(geog) FROM polygon_geog;
gid | ST_AsText
------------------------
 1 | POLYGON((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71))
(1 row)
```

Multipolygon

A collection of zero or more polygons that do not overlap.
Examples

The following example uses the GEOMETRY type to create a table, use copy to load a multipolygon into the table, and then queries the table to view the polygon:

```sql
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>9|MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
>>>.
=> SELECT gid, ST_AsText(geom) FROM polygon_geom;
gid | ST_AsText
----------------------
 9 | MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
(1 row)
```

The following example uses the GEOGRAPHY type to create a table, use copy to load a multipolygon into the table, and then queries the table to view the polygon:

```sql
=> CREATE TABLE multipolygon_geog (gid int, geom GEOGRAPHY(1000));
CREATE TABLE
=> COPY polygon_geog(gid, gx filler LONG VARCHAR, geom AS ST_GeographyFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>>1|POLYGON(((42.1 71, 41.4 70, 41.3 72.9, 44.47 73.21, 42.99 71.46, 42.1 71))
>>>.
=> SELECT gid, ST_AsText(geom) FROM polygon_geog;
gid | ST_AsText
----------------------
 1 | POLYGON(((42.1 71, 41.4 70, 41.3 72.9, 42.1 71)),((44.47 73.21, 42.99 71.46, 42.1 71, 44.47 73.21)))
(1 row)
```

Spatial Object Representations

The OGC defines two ways to represent spatial objects:

- Well-Known Text (WKT)
- Well-Known Binary (WKB)

Well-Known Text (WKT)

Well-Known Text (WKT) is an ASCII representation of a spatial object.
WKTs are not case sensitive; Vertica recognizes any combination of lowercase and uppercase letters.

Some examples of valid WKTs are:

<table>
<thead>
<tr>
<th>WKT Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT(1 2)</td>
<td>The point (1,2)</td>
</tr>
<tr>
<td>MULTIPOLYGON(0 0,1 1)</td>
<td>A set made up of the points (0,0) and (1,1)</td>
</tr>
<tr>
<td>LINESTRING(1.5 2.45,3.21 4)</td>
<td>The line from the point (1.5,2.45) to the point (3.21,4)</td>
</tr>
<tr>
<td>MULTILINESTRING((0 0,–1 –2,–3 –4),(2 3,3 4,6 7))</td>
<td>Two linestrings, one that passes through (0,0), (–1,–2), and (–3,–4), and one that passes through (2,3), (3,4), and (6,7).</td>
</tr>
<tr>
<td>POLYGON((1 2,1 4,3 4,3 2,1 2))</td>
<td>The rectangle whose four corners are indicated by (1,2), (1,4), (3,4), and (3,2). A polygon must be closed, so the first and last points in the WKT must match.</td>
</tr>
<tr>
<td>POLYGON((0.5 0.5,5 0.5 5,0.5 0.5), (1.5 1.4 3.4 1,1.5 1))</td>
<td>A polygon (0.5 0.5,5 0.5 5,0 5,0.5 0.5) with a hole in it (1.5 1.4 3.4 1,1.5 1).</td>
</tr>
<tr>
<td>MULTIPOLYGON(((0 1,3 0,4 3,0 4,0 1)), ((3 4,6 3,5 3,4)), ((0 0,–1 –2,–3 –2,–2 –1,0 0)))</td>
<td>A set of three polygons</td>
</tr>
<tr>
<td>GEOMETRYCOLLECTION(POINT (5 8), LINESTRING(–1 3,1 4))</td>
<td>A set containing the point (5,8) and the line from (–1,3) to (1,4)</td>
</tr>
<tr>
<td>POINT EMPTY</td>
<td>Empty spatial objects; empty objects have no points.</td>
</tr>
<tr>
<td>MULTIPOINT EMPTY</td>
<td></td>
</tr>
<tr>
<td>LINESTRING EMPTY</td>
<td></td>
</tr>
<tr>
<td>MULTILINESTRING EMPTY</td>
<td></td>
</tr>
<tr>
<td>MULTILINESTRING(EMPTY)</td>
<td></td>
</tr>
<tr>
<td>POLYGON EMPTY</td>
<td></td>
</tr>
<tr>
<td>POLYGON(EMPTY)</td>
<td></td>
</tr>
<tr>
<td>MULTIPOLYGON EMPTY</td>
<td></td>
</tr>
<tr>
<td>MULTIPOLYGON(EMPTY)</td>
<td></td>
</tr>
</tbody>
</table>

Invalid WKTs are:
- POINT(1 NaN), POINT(1 INF)—Coordinates must be numbers.
- POLYGON((1 2, 1 4, 3 4, 3 2))—A polygon must be closed.
- POLYGON((1 4, 2 4))—A linestring is not a valid polygon.

Well-Known Binary (WKB)

Well-Known Binary (WKB) is a binary representation of a spatial object. This format is primarily used to port spatial data between applications.

Spatial Definitions

The OGC defines properties that describe
- Characteristics of spatial objects
- Spatial relationships that can exist among objects

Vertica provides functions that test for and analyze the following properties and relationships.

Boundary

The set of points that define the limit of a spatial object:
- Points, multipoints, and geometrycollections do not have boundaries.
- The boundary of a linestring is a multipoint object. This object contains its start and end points.
- The boundary of a multilinestring is a multipoint object. This object contains the start and end points of all the linestrings that make up the multilinestring.
- The boundary of a polygon is a linestring that begins and ends at the same point. If the polygon has one or more holes, the boundary is a multilinestring that contains the boundaries of the exterior polygon and any interior polygons.
- The boundary of a multipolygon is a multilinestring that contains the boundaries of all the polygons that make up the multipolygon.


**Buffer**

The set of all points that are within or equal to a specified distance from the boundary of a spatial object. The distance can be positive or negative.

**Positive buffer:**

![Positive Buffer Diagram]

**Negative buffer:**

![Negative Buffer Diagram]

**Contains**

One spatial object contains another spatial object if its interior includes all points of the other object. If an object such as a point or linestring only exists along a polygon's boundary, the polygon does not contain it. If a point is on a linestring, the linestring contains it; the interior of a linestring is all the points on the linestring except the start and end points.

Contains(a, b) is spatially equivalent to within(b, a).

**Convex Hull**

The smallest convex polygon that contains one or more spatial objects.
In the following figure, the dotted lines represent the convex hull for a linestring and a triangle.

![Convex Hull Diagram](image)

**Crosses**

Two spatial objects cross if both of the following are true:

- The two objects have some but not all interior points in common.
- The dimension of the result of their intersection is less than the maximum dimension of the two objects.

![Crosses Example](image)

**Disjoint**

Two spatial objects have no points in common; they do not intersect or touch.

**Envelope**

The minimum bounding rectangle that contains a spatial object.
The envelope for the following polygon is represented by the dotted lines in the following figure.

Equals

Two spatial objects are equal when their coordinates match exactly. Synonymous with *spatially equivalent*.

The order of the points do not matter in determining spatial equivalence:

- LINESTRING(1 2, 4 3) equals LINESTRING(4 3, 1 2).
- POLYGON ((0 0, 1 1, 1 2, 2 2, 2 1, 3 0, 1.5 −1.5, 0 0)) equals POLYGON((1 1, 1 2, 2 2, 2 1, 3 0, 1.5 −1.5, 0 0, 1 1)).
- MULTILINESTRING((1 2, 4 3),(0 0, −1 −4)) equals MULTILINESTRING((0 0, −1 −4),(1 2, 4 3)).

Exterior

The set of points not contained within a spatial object nor on its boundary.

GeometryCollection

A set of zero or more objects from any of the supported classes of spatial objects.

Interior

The set of points contained in a spatial object, excluding its boundary.
Intersection

The set of points that two or more spatial objects have in common.

Overlaps

If a spatial object shares space with another object, but is not contained within that object, the objects overlap. The objects must overlap at their interiors; if two objects touch at a single point or intersect only along a boundary, they do not overlap.

Relates

When a spatial object is spatially related to another object as defined by a DE-9IM pattern matrix string.
A DE-9IM pattern matrix string identifies how two spatial objects are spatially related to each other. For more information about the DE-9IM standard, see Understanding Spatial Relations.

**Simple**

For points, multipoints, linestrings, or multilinestrings, a spatial object is simple if it does not intersect itself or has no self-tangency points.

Polygons, multipolygons, and geometrycollections are always simple.

**Symmetric Difference**

The set of all points of a pair of spatial objects where the objects do not intersect. This difference is spatially equivalent to the union of the two objects less their intersection. The symmetric difference contains the boundaries of the intersections.

In the following figure, the shaded areas represent the symmetric difference of the two rectangles.

The following figure shows the symmetric difference of two overlapping linestrings.
Union

For two or more spatial objects, the set of all points in all the objects.

Validity

For a polygon or multipolygon, when all of the following are true:

- It is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.
- No two linestrings in the boundary cross. The linestrings in the boundary may touch at a point but they cannot cross.

- Any polygons in the interior must be completely contained; they cannot touch the boundary of the exterior polygon except at a vertex.

Valid polygons:

![Valid Polygon Diagram]

Invalid polygon:

![Invalid Polygon Diagram]

Within

A spatial object is considered within another spatial object when all its points are inside the other object's interior. Thus, if a point or linestring only exists along a polygon's boundary, it is not considered within the polygon. The polygon boundary is not part of its interior.

If a point is on a linestring, it is considered within the linestring. The interior of a linestring is all the points along the linestring, except the start and end points.

Within(a, b) is spatially equivalent to contains(b, a).
Spatial Data Type Support Limitations

Vertica does not support all types of GEOMETRY and GEOGRAPHY objects. See the respective function page for a list of objects that function supports. Spherical geometry is generally more complex than Euclidean geometry. Thus, there are fewer spatial functions that support the GEOGRAPHY data type.

Limitations of spatial data type support:

- A non-WGS84 GEOGRAPHY object is a spatial object defined on the surface of a perfect sphere of radius 6371 kilometers. This sphere approximates the shape of the earth. Other spatial programs may use an ellipsoid to model the earth, resulting in slightly different data.

- You cannot modify the size or data type of a GEOMETRY or GEOGRAPHY column after creation.

- You cannot import data to or export data from tables that contain spatial data from another Vertica database.

- You can only use the STV_Intersect functions with points and polygons.

- GEOGRAPHY objects of type GEOMETRYCOLLECTION are not supported.

- Values for longitude must be between -180 and +180 degrees. Values for latitude must be between –90 and +90 degrees. The Vertica geospatial functions do not validate these values.

- GEOMETRYCOLLECTION objects cannot contain empty objects. For example, you cannot specify GEOMETRYCOLLECTION (LINESTRING(1 2, 3 4), POINT(5 6), POINT EMPTY).

- If you pass a spatial function a NULL geometry, the function returns NULL, unless otherwise specified. A result of NULL has no value.
Polymorphic functions, such as NVL and GREATEST, do not accept GEOMETRY and GEOGRAPHY arguments.
Time Series Analytics

Time series analytics evaluate the values of a given set of variables over time and group those values into a window (based on a time interval) for analysis and aggregation. Common scenarios for using time series analytics include: stock market trades and portfolio performance changes over time, and charting trend lines over data.

Since both time and the state of data within a time series are continuous, it can be challenging to evaluate SQL queries over time. Input records often occur at non-uniform intervals, which can create gaps. To solve this problem Vertica provides:

- Gap-filling functionality, which fills in missing data points
- Interpolation scheme, which constructs new data points within the range of a discrete set of known data points.

Vertica interpolates the non-time series columns in the data (such as analytic function results computed over time slices) and adds the missing data points to the output. This section describes gap filling and interpolation in detail.

You can use event-based windows to break time series data into windows that border on significant events within the data. This is especially relevant in financial data, where analysis might focus on specific events as triggers to other activity.

Sessionization is a special case of event-based windows that is frequently used to analyze click streams, such as identifying web browsing sessions from recorded web clicks.

Vertica provides additional support for time series analytics with the following SQL extensions:

- TIMESERIES clause in a SELECT statement supports gap-filling and interpolation (GFI) computation.
- TS_FIRST_VALUE and TS_LAST_VALUE are time series aggregate functions that return the value at the start or end of a time slice, respectively, which is determined by the interpolation scheme.
- TIME_SLICE is a (SQL extension) date/time function that aggregates data by different fixed-time intervals and returns a rounded-up input TIMESTAMP value to a value that corresponds with the start or end of the time slice interval.
See Also

- SQL Analytics
- Event-Based Windows
- Sessionization with Event-Based Windows
Gap Filling and Interpolation (GFI)

The examples and graphics that explain the concepts in this topic use the following simple schema:

```sql
CREATE TABLE TickStore (ts TIMESTAMP, symbol VARCHAR(8), bid FLOAT);
INSERT INTO TickStore VALUES ('2009-01-01 03:00:00', 'XYZ', 10.0);
INSERT INTO TickStore VALUES ('2009-01-01 03:00:05', 'XYZ', 10.5);
COMMIT;
```

In Vertica, time series data is represented by a sequence of rows that conforms to a particular table schema, where one of the columns stores the time information.

Both time and the state of data within a time series are continuous. Thus, evaluating SQL queries over time can be challenging because input records usually occur at non-uniform intervals and can contain gaps.

For example, the following table contains two input rows five seconds apart: 3:00:00 and 3:00:05.

```
=> SELECT * FROM TickStore;
```

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:05</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(2 rows)

Given those two inputs, how can you determine a bid price that falls between the two points, such as at 3:00:03 PM?

The `TIME_SLICE` function normalizes timestamps into corresponding time slices; however, `TIME_SLICE` does not solve the problem of missing inputs (time slices) in the data. Instead, Vertica provides gap-filling and interpolation (GFI) functionality, which fills in missing data points and adds new (missing) data points within a range of known data points to the output. It accomplishes these tasks with time series aggregate functions and the SQL TIMESERIES Clause.

But first, we'll illustrate the components that make up gap filling and interpolation in Vertica, starting with Constant Interpolation.

The images in the following topics use the following legend:

- The x-axis represents the timestamp (`ts`) column
- The y-axis represents the bid column.
- The vertical blue lines delimit the time slices.
The red dots represent the input records in the table, $10.0$ and $10.5$.

The blue stars represent the output values, including interpolated values.

**Constant Interpolation**

Given known input timestamps at 03:00:00 and 03:00:05 in the sample TickStore schema, how might you determine the bid price at 03:00:03?

A common interpolation scheme used on financial data is to set the bid price to the last seen value so far. This scheme is referred to as constant interpolation, in which Vertica computes a new value based on the previous input records.

*Note: Constant is Vertica’s default interpolation scheme. Another interpolation scheme, linear, is discussed in an upcoming topic.*

Returning to the problem query, here is the table output, which shows a 5-second lag between bids at 03:00:00 and 03:00:05:

```sql
=> SELECT * FROM TickStore;
<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:05</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>
```

(2 rows)

Using constant interpolation, the interpolated bid price of XYZ remains at $10.0$ at 3:00:03, which falls between the two known data inputs (3:00:00 PM and 3:00:05). At 3:00:05, the value changes to $10.5$. The known data points are represented by a red dot, and the interpolated value at 3:00:03 is represented by the blue star.

In order to write a query that makes the input rows more uniform, you first need to understand the TIMESERIES clause and time series aggregate functions.
TIMESERIES Clause and Aggregates

The SELECT..TIMESERIES clause and time series aggregates help solve the problem of gaps in input records by normalizing the data into 3-second time slices and interpolating the bid price when it finds gaps.

TIMESERIES Clause

The TIMESERIES Clause is an important component of time series analytics computation. It performs gap filling and interpolation (GFI) to generate time slices missing from the input records. The clause applies to the timestamp columns/expressions in the data, and takes the following form:

```
TIMESERIES slice_time AS 'length_and_time_unit_expression'
OVER ( ... [ window-partition-clause[ , ... ] ]
... ORDER BY time_expression )
... [ ORDER BY table_column [ , ... ] ]
```

Note: The TIMESERIES clause requires an ORDER BY operation on the timestamp column.

Time Series Aggregate Functions

Timeseries Aggregate (TSA) functions evaluate the values of a given set of variables over time and group those values into a window for analysis and aggregation.

TSA functions process the data that belongs to each time slice. One output row is produced per time slice or per partition per time slice if a partition expression is present.

The following table shows 3-second time slices where:

- The first two rows fall within the first time slice, which runs from 3:00:00 to 3:00:02. These are the input rows for the TSA function's output for the time slice starting at 3:00:00.

- The second two rows fall within the second time slice, which runs from 3:00:03 to 3:00:05. These are the input rows for the TSA function's output for the time slice starting at 3:00:03.
The result is the start of each time slice.

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10.0</td>
</tr>
<tr>
<td>2009-01-01 03:00:01</td>
<td>XYZ</td>
<td>10.1</td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td>10.3</td>
</tr>
<tr>
<td>2009-01-01 03:00:05</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10.0</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>10.1</td>
</tr>
</tbody>
</table>

Examples

The following examples compare the values returned with and without the TS_FIRST_VALUE TSA function.

This example shows the TIMESERIES clause without the TS_FIRST_VALUE TSA function.

```sql
=> SELECT slice_time, bid FROM TickStore TIMESERIES slice_time AS '3 seconds' OVER(PARTITION by TickStore.bid ORDER BY ts);
```

This example shows both the TIMESERIES clause and the TS_FIRST_VALUE TSA function. The query returns the values of the bid column, as determined by the specified constant interpolation scheme.

```sql
=> SELECT slice_time, TS_FIRST_VALUE(bid, 'CONST') bid FROM TickStore TIMESERIES slice_time AS '3 seconds' OVER(PARTITION by symbol ORDER BY ts);
```

Vertica interpolates the last known value and fills in the missing datapoint, returning 10 at 3:00:03:

<table>
<thead>
<tr>
<th>First query</th>
<th>Interpolated value</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice_time</td>
<td>bid</td>
</tr>
<tr>
<td>2009-01-01 03:00:00</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(2 rows)

Time Series Rounding

Vertica calculates all time series as equal intervals relative to the timestamp 2000-01-01 00:00:00. Vertica rounds time series timestamps as needed, to conform with this baseline. Start times are also rounded down to the nearest whole unit for the specified interval.

Given this logic, the TIMESERIES clause generates series of timestamps as described in the following sections.
Minutes

Time series of minutes are rounded down to full minutes. For example, the following statement specifies a time span of 00:00:03 - 00:05:50:

```sql
=> SELECT ts FROM ( 
    SELECT '2015-01-04 00:00:03'::TIMESTAMP AS tm 
    UNION 
    SELECT '2015-01-04 00:05:50'::TIMESTAMP AS tm 
  ) t 
TIMESERIES ts AS '1 minute' OVER (ORDER BY tm);
```

Vertica rounds down the time series start and end times to full minutes, 00:00:00 and 00:05:00, respectively:

```
   ts
  -------
2015-01-04 00:00:00
2015-01-04 00:01:00
2015-01-04 00:02:00
2015-01-04 00:03:00
2015-01-04 00:04:00
2015-01-04 00:05:00
(6 rows)
```

Weeks

Because the baseline timestamp 2000-01-01 00:00:00 is a Saturday, all time series of weeks start on Saturday. Vertica rounds down the series start and end timestamps accordingly. For example, the following statement specifies a time span of 12/10/99 - 01/10/00:

```sql
=> SELECT ts FROM ( 
    SELECT '1999-12-10 00:00:00'::TIMESTAMP AS tm 
    UNION 
    SELECT '2000-01-10 23:59:59'::TIMESTAMP AS tm 
  ) t 
TIMESERIES ts AS '1 week' OVER (ORDER BY tm);
```

The specified time span starts on Friday (12/10/99), so Vertica starts the time series on the preceding Saturday, 12/04/99. The time series ends on the last Saturday within the time span, 01/08/00:

```
   ts
  -------
1999-12-04 00:00:00
1999-12-11 00:00:00
1999-12-18 00:00:00
```

Months

Time series of months are divided into equal 30-day intervals, relative to the baseline timestamp 2000-01-01 00:00:00. For example, the following statement specifies a time span of 09/01/99 - 12/31/00:

```sql
=> SELECT ts FROM (
    SELECT '1999-09-01 00:00:00'::TIMESTAMP AS tm
    UNION
    SELECT '2000-12-31 23:59:59'::TIMESTAMP AS tm
) t
   TIMESERIES ts AS '1 month' OVER (ORDER BY tm);
```

Vertica generates a series of 30-day intervals, where each timestamp is rounded up or down, relative to the baseline timestamp:

<table>
<thead>
<tr>
<th>ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999-08-04</td>
</tr>
<tr>
<td>1999-09-03</td>
</tr>
<tr>
<td>1999-10-03</td>
</tr>
<tr>
<td>1999-11-02</td>
</tr>
<tr>
<td>1999-12-02</td>
</tr>
<tr>
<td>2000-01-01</td>
</tr>
<tr>
<td>2000-01-31</td>
</tr>
<tr>
<td>2000-03-01</td>
</tr>
<tr>
<td>2000-03-31</td>
</tr>
<tr>
<td>2000-04-30</td>
</tr>
<tr>
<td>2000-05-30</td>
</tr>
<tr>
<td>2000-06-29</td>
</tr>
<tr>
<td>2000-07-29</td>
</tr>
<tr>
<td>2000-08-28</td>
</tr>
<tr>
<td>2000-09-27</td>
</tr>
<tr>
<td>2000-10-27</td>
</tr>
<tr>
<td>2000-11-26</td>
</tr>
<tr>
<td>2000-12-26</td>
</tr>
</tbody>
</table>

(18 rows)

Years

Time series of years are divided into equal 365-day intervals. If a time span overlaps leap years since or before the baseline timestamp 2000-01-01 00:00:00, Vertica rounds the series timestamps accordingly.
For example, the following statement specifies a time span of 01/01/95 - 05/08/09, which overlaps four leap years, including the baseline timestamp:

```sql
=> SELECT ts FROM (  
    SELECT '1995-01-01 00:00:00'::TIMESTAMP AS tm  
    UNION  
    SELECT '2009-05-08'::TIMESTAMP AS tm  
) t timeseries ts AS '1 year' over (ORDER BY tm);
```

Vertica generates a series of timestamps that are rounded up or down, relative to the baseline timestamp:

```
t    
1994-01-02 00:00:00
1995-01-02 00:00:00
1996-01-02 00:00:00
1997-01-01 00:00:00
1998-01-01 00:00:00
1999-01-01 00:00:00
2000-01-01 00:00:00
2000-12-31 00:00:00
2001-12-31 00:00:00
2002-12-31 00:00:00
2003-12-31 00:00:00
2004-12-30 00:00:00
2005-12-30 00:00:00
2006-12-30 00:00:00
2007-12-30 00:00:00
2008-12-29 00:00:00
(16 rows)
```

### Linear Interpolation

Instead of interpolating data points based on the last seen value (Constant Interpolation), linear interpolation is where Vertica interpolates values in a linear slope based on the specified time slice.

The query that follows uses linear interpolation to place the input records in 2-second time slices and return the first bid value for each symbol/time slice combination (the value at the start of the time slice):

```sql
=> SELECT slice_time, TS_FIRST_VALUE(bid, 'LINEAR') bid FROM Tickstore  
    TIMESERIES slice_time AS '2 seconds' OVER(PARTITION BY symbol ORDER BY ts);
```

```
slice_time | bid  
-----------
2009-01-01 03:00:00 | 10
2009-01-01 03:00:02 | 10.2
2009-01-01 03:00:04 | 10.4
(3 rows)
```
The following figure illustrates the previous query results, showing the 2-second time gaps (3:00:02 and 3:00:04) in which no input record occurs. Note that the interpolated bid price of XYZ changes to 10.2 at 3:00:02 and 10.3 at 3:00:03 and 10.4 at 3:00:04, all of which fall between the two known data inputs (3:00:00 and 3:00:05). At 3:00:05, the value would change to 10.5.

### Note

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:00:00</td>
<td>XYZ</td>
<td>10.0</td>
</tr>
<tr>
<td>3:00:02</td>
<td>XYZ</td>
<td>10.2</td>
</tr>
<tr>
<td>3:00:04</td>
<td>XYZ</td>
<td>10.4</td>
</tr>
<tr>
<td>3:00:05</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>

The following is a side-by-side comparison of constant and linear interpolation schemes.
GFI Examples

This topic illustrates some of the queries you can write using the constant and linear interpolation schemes.

Constant Interpolation

The first query uses TS_FIRST_VALUE() and the TIMESERIES Clause to place the input records in 3-second time slices and return the first bid value for each symbol/time slice combination (the value at the start of the time slice).

Note: The TIMESERIES clause requires an ORDER BY operation on the TIMESTAMP column.

```sql
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid) AS first_bid FROM TickStore
    TIMESERIES slice_time AS '3 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

Because the bid price of stock XYZ is 10.0 at 3:00:03, the first_bid value of the second time slice, which starts at 3:00:03 is till 10.0 (instead of 10.5) because the input value of 10.5 does not occur until 3:00:05. In this case, the interpolated value is inferred from the last value seen on stock XYZ for time 3:00:03:

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>first_bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>(2 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The next example places the input records in 2-second time slices to return the first bid value for each symbol/time slice combination:

```sql
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid) AS first_bid FROM TickStore
    TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

The result now contains three records in 2-second increments, all of which occur between the first input row at 03:00:00 and the second input row at 3:00:05. Note that the second and third output record correspond to a time slice where there is no input record:

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>first_bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>(3 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using the same table schema, the next query uses \texttt{TS\_LAST\_VALUE()}, with the \texttt{TIMESERIES} clause to return the last values of each time slice (the values at the end of the time slices).

\textbf{Note:} Time series aggregate functions process the data that belongs to each time slice. One output row is produced per time slice or per partition per time slice if a partition expression is present.

\begin{verbatim}
=> SELECT slice_time, symbol, TS\_LAST\_VALUE(bid) AS last_bid FROM TickStore
    TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
\end{verbatim}

Notice that the last value output row is 10.5 because the value 10.5 at time 3:00:05 was the last point inside the 2-second time slice that started at 3:00:04:

\begin{verbatim}
slice_time | symbol | last_bid
-------------|--------|--------
2009-01-01 03:00:00 | XYZ    | 10.2
2009-01-01 03:00:02 | XYZ    | 10.4
2009-01-01 03:00:04 | XYZ    | 10.5
(3 rows)
\end{verbatim}

Remember that because constant interpolation is the default, the same results are returned if you write the query using the \texttt{CONST} parameter as follows:

\begin{verbatim}
=> SELECT slice_time, symbol, TS\_LAST\_VALUE(bid, 'CONST') AS last_bid FROM TickStore
    TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
\end{verbatim}

**Linear Interpolation**

Based on the same input records described in the constant interpolation examples, which specify 2-second time slices, the result of \texttt{TS\_LAST\_VALUE} with linear interpolation is as follows:

\begin{verbatim}
=> SELECT slice_time, symbol, TS\_LAST\_VALUE(bid, 'linear') AS last_bid FROM TickStore
    TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
\end{verbatim}

In the results, no \texttt{last\_bid} value is returned for the last row because the query specified \texttt{TS\_LAST\_VALUE}, and there is no data point after the 3:00:04 time slice to interpolate.

\begin{verbatim}
slice_time | symbol | last_bid
-------------|--------|--------
2009-01-01 03:00:00 | XYZ    | 10.2
2009-01-01 03:00:02 | XYZ    | 10.4
2009-01-01 03:00:04 | XYZ    |
(3 rows)
\end{verbatim}
Using Multiple Time Series Aggregate Functions

Multiple time series aggregate functions can exists in the same query. They share the same gap-filling policy as defined in the TIMESERIES clause; however, each time series aggregate function can specify its own interpolation policy. In the following example, there are two constant and one linear interpolation schemes, but all three functions use a three-second time slice:

```sql
=> SELECT slice_time, symbol,
   TS_FIRST_VALUE(bid, 'const') fv_c,
   TS_FIRST_VALUE(bid, 'linear') fv_l,
   TS_LAST_VALUE(bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS '3 seconds' OVER(PARTITION BY symbol ORDER BY ts);
```

In the following output, the original output is compared to output returned by multiple time series aggregate functions.

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>=&gt;</th>
<th>slice_time</th>
<th>symbol</th>
<th>fv_c</th>
<th>fv_l</th>
<th>lv_c</th>
</tr>
</thead>
<tbody>
<tr>
<td>03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td></td>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>03:00:05</td>
<td>XYZ</td>
<td>10.5</td>
<td></td>
<td>2009-01-01 03:00:03</td>
<td>XYZ</td>
<td>10</td>
<td>10.3</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(2 rows)

Using the Analytic LAST_VALUE Function

Here’s an example using LAST_VALUE(), so you can see the difference between it and the GFI syntax.

```sql
=> SELECT *, LAST_VALUE(bid) OVER(PARTITION by symbol ORDER BY ts) AS "last bid" FROM TickStore;
```

There is no gap filling and interpolation to the output values.

<table>
<thead>
<tr>
<th>ts</th>
<th>symbol</th>
<th>bid</th>
<th>last bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>03:00:00</td>
<td>XYZ</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>03:00:05</td>
<td>XYZ</td>
<td>10.5</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(2 rows)

Using slice_time

In a TIMESERIES query, you cannot use the column slice_time in the WHERE clause because the WHERE clause is evaluated before the TIMESERIES clause, and the slice_time column is
not generated until the TIMESERIES clause is evaluated. For example, Vertica does not support the following query:

```sql
=> SELECT symbol, slice_time, TS_FIRST_VALUE(bid IGNORE NULLS) AS fv
   FROM TickStore
   WHERE slice_time = '2009-01-01 03:00:00'
   TIMESERIES slice_time as '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
ERROR: Time Series timestamp alias/Time Series Aggregate Functions not allowed in WHERE clause
```

Instead, you could write a subquery and put the predicate on `slice_time` in the outer query:

```sql
=> SELECT * FROM (SELECT symbol, slice_time,
   TS_FIRST_VALUE(bid IGNORE NULLS) AS fv
   FROM TickStore
   TIMESERIES slice_time AS '2 seconds'
   OVER (PARTITION BY symbol ORDER BY ts) ) sq
   WHERE slice_time = '2009-01-01 03:00:00';
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>slice_time</th>
<th>fv</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>2009-01-01 03:00:00</td>
<td>10</td>
</tr>
</tbody>
</table>

(1 row)

Creating a Dense Time Series

The TIMESERIES clause provides a convenient way to create a dense time series for use in an outer join with fact data. The results represent every time point, rather than just the time points for which data exists.

The examples that follow use the same TickStore schema described in Gap Filling and Interpolation (GFI), along with the addition of a new inner table for the purpose of creating a join:

```sql
=> CREATE TABLE inner_table (ts TIMESTAMP, bid FLOAT);
=> CREATE PROJECTION inner_p (ts, bid) as SELECT * FROM inner_table ORDER BY ts, bid UNSEGMENTED ALL NODES;
=> INSERT INTO inner_table VALUES ('2009-01-01 03:00:02', 1);
=> INSERT INTO inner_table VALUES ('2009-01-01 03:00:04', 2);
=> COMMIT;
```

You can create a simple union between the start and end range of the timeframe of interest in order to return every time point. This example uses a 1-second time slice:

```sql
=> SELECT ts FROM (SELECT '2009-01-01 03:00:00':TIMESTAMP AS time FROM TickStore
   UNION
   SELECT '2009-01-01 03:00:05':TIMESTAMP FROM TickStore) t
   TIMESERIES ts AS '1 seconds' OVER(ORDER BY time);
```
The next query creates a union between the start and end range of the timeframe using 500-millisecond time slices:

```sql
=> SELECT ts FROM ( 
    SELECT '2009-01-01 03:00:00'::TIMESTAMP AS time 
    FROM TickStore 
    UNION 
    SELECT '2009-01-01 03:00:05'::TIMESTAMP FROM TickStore) t 
TIMESERIES ts AS '500 milliseconds' OVER(ORDER BY time);
```

<table>
<thead>
<tr>
<th>ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
</tr>
<tr>
<td>2009-01-01 03:00:01</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
</tr>
<tr>
<td>2009-01-01 03:00:03</td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
</tr>
<tr>
<td>2009-01-01 03:00:05</td>
</tr>
</tbody>
</table>

(6 rows)

The following query creates a union between the start- and end-range of the timeframe of interest using 1-second time slices:

```sql
=> SELECT * FROM ( 
    SELECT ts FROM ( 
        SELECT '2009-01-01 03:00:00'::timestamp AS time FROM TickStore 
        UNION 
        SELECT '2009-01-01 03:00:05'::timestamp FROM TickStore) t 
    TIMESERIES ts AS '1 seconds' OVER(ORDER BY time) ) AS outer_table 
LEFT OUTER JOIN inner_table ON outer_table.ts = inner_table.ts;
```

The union returns a complete set of records from the left-joined table with the matched records in the right-joined table. Where the query found no match, it extends the right side column with null values:

<table>
<thead>
<tr>
<th>ts</th>
<th>ts</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-01-01 03:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>2009-01-01 03:00:02</td>
<td>1</td>
</tr>
</tbody>
</table>
Null Values in Time Series Data

Null values are not common inputs for gap-filling and interpolation (GFI) computation, but if null values do exist, you can use time series aggregate functions (TS_FIRST_VALUE/TS_LAST_VALUE) with the IGNORE NULLS arguments to affect output of the interpolated values. The TSA functions are treated similarly to their analytic counterparts (FIRST_VALUE/LAST_VALUE) in that if the timestamp itself is null Vertica filter out those rows before gap filling and interpolation occurs.

The three images below will illustrate the points that follow on how Vertica handles time series data that contains null values.

| Figure 1. Interpolated bid values when the input has no NULLs | Figure 2. CONST-interpolated bid values when the input has NULL values | Figure 3. LINEAR-interpolated bid values when the input has NULL values |

![Figure 1](image1.png) ![Figure 2](image2.png) ![Figure 3](image3.png)
Constant Interpolation with Null Values

Figure 1 illustrates a default (constant) interpolation result on four input rows where none of the inputs contains a NULL value. Figure 2 shows the same input rows with the addition of another input record whose bid value is NULL, and whose timestamp (ts) value is 3:00:03.

For constant interpolation, the bid value starting at 3:00:03 is null until the next non-null bid value appears in time. In Figure 2, the presence of the null row makes the interpolated bid value null in the time interval denoted by the shaded region. As a result, if TS_FIRST_VALUE(bid) is evaluated with constant interpolation on the time slice that begins at 3:00:02, its output is non-null. However, TS_FIRST_VALUE(bid) on the next time slice produces null. If the last value of the 3:00:02 time slice is null, the first value for the next time slice (3:00:04) is null. However, if you were to use a TSA function with IGNORE NULLS, then the value at 3:00:04 would be the same value as it was at 3:00:02.

To illustrate, insert a new row into the TickStore table at 03:00:03 with a null bid value, Vertica will output a row for the 03:00:02 record with a null value but no row for the 03:00:03 input:

```
=> INSERT INTO tickstore VALUES('2009-01-01 03:00:03', 'XYZ', NULL);
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid) AS last_bid FROM TickStore
  -> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>last_bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>XYZ</td>
<td>NULL</td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>
(3 rows)
```

If you specify IGNORE NULLS, Vertica fills in the missing data point using a constant interpolation scheme. Here, the bid price at 03:00:02 is interpolated to the last known input record for bid, which was $10 at 03:00:00:

```
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid IGNORE NULLS) AS last_bid FROM TickStore
  -> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>last_bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>
(3 rows)
```

Now if you were to insert a row where the timestamp column contained a null value, Vertica would filter out that row before gap filling and interpolation occurred.

```
=> INSERT INTO tickstore VALUES(NULL, 'XYZ', 11.2);
=> SELECT slice_time, symbol, TS_LAST_VALUE(bid) AS last_bid FROM TickStore
  -> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```
Notice there is no output for the 11.2 bid row:

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>last_bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>XYZ</td>
<td></td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td>10.5</td>
</tr>
</tbody>
</table>

(3 rows)

Linear Interpolation with Null Values

For linear interpolation, the interpolated bid value becomes null in the time interval, which is represented by the shaded region in Figure 3. In the presence of an input null value at 3:00:03, Vertica cannot linearly interpolate the bid value around that time point.

Vertica takes the closest non null value on either side of the time slice and uses that value. For example, if you use a linear interpolation scheme and you do not specify IGNORE NULLS, and your data has one real value and one null, the result is null. If the value on either side is null, the result is null. Therefore, to evaluate TS_FIRST_VALUE(bid) with linear interpolation on the time slice that begins at 3:00:02, its output is null. TS_FIRST_VALUE(bid) on the next time slice remains null.

```sql
=> SELECT slice_time, symbol, TS_FIRST_VALUE(bid, 'linear') AS fv_1 FROM TickStore
-> TIMESERIES slice_time AS '2 seconds' OVER (PARTITION BY symbol ORDER BY ts);
```

<table>
<thead>
<tr>
<th>slice_time</th>
<th>symbol</th>
<th>fv_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-01-01 03:00:00</td>
<td>XYZ</td>
<td>10</td>
</tr>
<tr>
<td>2009-01-01 03:00:02</td>
<td>XYZ</td>
<td></td>
</tr>
<tr>
<td>2009-01-01 03:00:04</td>
<td>XYZ</td>
<td></td>
</tr>
</tbody>
</table>

(3 rows)
Data Aggregation

You can use functions such as `SUM` and `COUNT` to aggregate the results of GROUP BY queries at one or more levels.

Aggregating Data at a Single Level

The simplest GROUP BY queries aggregate data at a single level. For example, a table might contain the following information about family expenses:

- Category
- Amount spent on that category during the year
- Year

Table data might look like this:

```sql
=> SELECT * FROM expenses ORDER BY Category;
Year | Category | Amount
-----|----------|--------
2005 | Books    | 39.98  
2007 | Books    | 29.99  
2008 | Books    | 29.99  
2006 | Electrical | 109.99 
2005 | Electrical | 109.99 
2007 | Electrical | 229.98 
```

You can use aggregate functions to get the total expenses per category or per year:

```sql
=> SELECT SUM(Amount), Category FROM expenses GROUP BY Category;
SUM   | Category
-----|----------
99.96 | Books    
449.96| Electrical

=> SELECT SUM(Amount), Year FROM expenses GROUP BY Year;
SUM   | Year
-----|-----
149.97| 2005
109.99| 2006
29.99 | 2008
259.97| 2007
```
Aggregating Data at Multiple Levels

Over time, tables that are updated frequently can contain large amounts of data. Using the simple table shown earlier, suppose you want a multilevel query, like the number of expenses per category per year.

The following query uses the ROLLUP aggregation with the SUM function to calculate the total expenses by category and the overall expenses total. The NULL fields indicate subtotal values in the aggregation.

- When only the Year column is NULL, the subtotal is for all the Category values.
- When both the Year and Category columns are NULL, the subtotal is for all Amount values for both columns.

Using the ORDER BY clause orders the results by expense category, the year the expenses took place, and the GROUP BY level that the GROUPING_ID function creates:

```sql
=> SELECT Category, Year, SUM(Amount) FROM expenses
   GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electrical</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Electrical</td>
<td></td>
<td>449.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
</tr>
</tbody>
</table>

Similarly, the following query calculates the total sales by year and the overall sales total and then uses the ORDER BY clause to sort the results:

```sql
=> SELECT Category, Year, SUM(Amount) FROM expenses
   GROUP BY ROLLUP(Year, Category) ORDER BY 2, 1, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Electrical</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2006</td>
<td>149.97</td>
</tr>
<tr>
<td>Electrical</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2007</td>
<td>259.97</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
</tbody>
</table>
You can use the CUBE aggregate to perform all possible groupings of the category and year expenses. The following query returns all possible groupings, ordered by grouping:

```sql
=> SELECT Category, Year, SUM(Amount) FROM expenses
    GROUP BY CUBE(Category, Year) ORDER BY 1, 2, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electrical</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Electrical</td>
<td></td>
<td>449.96</td>
</tr>
<tr>
<td></td>
<td>2005</td>
<td>149.97</td>
</tr>
<tr>
<td></td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td></td>
<td>2007</td>
<td>259.97</td>
</tr>
<tr>
<td></td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
</tr>
</tbody>
</table>

The results include subtotals for each category and each year and a total ($549.92) for all transactions, regardless of year or category.

ROLLUP, CUBE, and GROUPING SETS generate NULL values in grouping columns to identify subtotals. If table data includes NULL values, differentiating these from NULL values in subtotals can sometimes be challenging.

In the preceding output, the NULL values in the Year column indicate that the row was grouped on the Category column, rather than on both columns. In this case, ROLLUP added the NULL value to indicate the subtotal row.

To distinguish subtotal rows from NULL values that are part of the input data, use the GROUPING function.

## Aggregate Expressions for GROUP BY

You can include CUBE and ROLLUP aggregates within a GROUPING SETS aggregate. Be aware that the CUBE and ROLLUP aggregates can result in a large amount of output. However, you can avoid large outputs by using GROUPING SETS to return only specified results.

```sql
...GROUP BY a,b,c,d,ROLLUP(a,b)...
...GROUP BY a,b,c,d,CUBE((a,b),c,d)...
```

You cannot include any aggregates in a CUBE or ROLLUP aggregate expression.
You can append multiple GROUPING SETS, CUBE, or ROLLUP aggregates in the same query.

```sql
...GROUP BY a,b,c,d,CUBE(a,b),ROLLUP (c,d)...
...GROUP BY a,b,c,d,GROUPING SETS ((a,d),(b,c),CUBE(a,b))...
...GROUP BY a,b,c,d,GROUPING SETS ((a,d),(b,c),(a,b),(a),(b),(()))...
```

## Aggregates and Functions for Multilevel Grouping

Vertica provides several aggregates and functions that group the results of a GROUP BY query at multiple levels.

### Aggregates for Multilevel Grouping

Use the following aggregates for multilevel grouping:

- **ROLLUP** automatically performs subtotal aggregations. ROLLUP performs one or more aggregations across multiple dimensions, at different levels.

- **CUBE** performs the aggregation for all permutations of the CUBE expression that you specify.

- **GROUPING SETS** let you specify which groupings of aggregations you need.

You can use CUBE or ROLLUP expressions inside GROUPING SETS expressions. Otherwise, you cannot nest multilevel aggregate expressions.

### Grouping Functions

You use one of the following three grouping functions with ROLLUP, CUBE, and GROUPING SETS:

- **GROUP_ID** returns one or more numbers, starting with zero (0), to uniquely identify duplicate sets.

- **GROUPING_ID** produces a unique ID for each grouping combination.
GROUPING identifies for each grouping combination whether a column is a part of this grouping. This function also differentiates NULL values in the data from NULL grouping subtotals.

These functions are typically used with multilevel aggregates.

Pre-Aggregating Data in Projections

Queries that use aggregate functions such as SUM and COUNT can perform more efficiently when they use projections that already contain the aggregated data. This improved efficiency is especially true for queries on large quantities of data.

For example, a power grid company reads 30 million smart meters that provide data at five-minute intervals. The company records each reading in a database table. Over a given year, three trillion records are added to this table.

The power grid company can analyze these records with queries that include aggregate functions to perform the following tasks:

- Establish usage patterns.
- Detect fraud.
- Measure correlation to external events such as weather patterns or pricing changes.

To optimize query response time, you can create an aggregate projection, which stores the data is stored after it is aggregated.

Aggregate Projections

Vertica provides three types of projections for storing data that is returned from aggregate functions or expressions:

- Projection that contains expressions: Projection with columns whose values are calculated from anchor table columns.
- Live aggregate projection: Projection that contains columns with values that are aggregated from columns in its anchor table. You can also define live aggregate projections that include user-defined transform functions.
- **Top-K projection**: Type of live aggregate projection that returns the top $k$ rows from a partition of selected rows. Create a Top-K projection that satisfies the criteria for a Top-K query.

### Recommended Use
- Aggregate projections are most useful for queries against large sets of data.
- For optimal query performance, the size of LAP projections should be a small subset of the anchor table—ideally, between 1 and 10 percent of the anchor table, or smaller, if possible.
- Because you cannot update or delete data in tables that use aggregate projections, use aggregate projections for tables whose data is cumulative and require infrequent updates and deletions.

### Requirements
- To use aggregate projections, you must set the appropriate configuration parameters (see [Enabling Live Aggregate Projections and Projections with Expressions]).
- In the event of manual recovery from an unclean database shutdown, live aggregate projections might require some time before they are refreshed.

### Enabling Live Aggregate Projections and Projections with Expressions
To create live aggregate and Top-K projections, or projections with expressions, the following configuration parameters must be set to 1 (default setting):

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Description</th>
<th>Enable</th>
<th>Disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableGroupByProjections</td>
<td>Enables live aggregate projections</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EnableTopKProjections</td>
<td>Enables Top-K projections</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EnableExprsInProjections</td>
<td>Enables projections with expressions.</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
To check current settings, use `SHOW CURRENT`. For example:

```sql
=> SHOW CURRENT EnableGroupByProjections;

<table>
<thead>
<tr>
<th>level</th>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>EnableGroupByProjections</td>
<td>1</td>
</tr>
</tbody>
</table>
(1 row)
```

**Live Aggregate Projections**

A live aggregate projection contains columns with values that are aggregated from columns in its anchor table. When you load data into the table, Vertica aggregates the data before loading it into the live aggregate projection. On subsequent loads—for example, through `INSERT` or `COPY`—Vertica recalculates aggregations with the new data and updates the projection.

**Functions Supported for Live Aggregate Projections**

Vertica can aggregate results in live aggregate projections from the following aggregate functions:

- `SUM`
- `MAX`
- `MIN`
- `COUNT`

**Aggregate Functions with DISTINCT**

Live aggregate projections can support queries that include aggregate functions qualified with the keyword `DISTINCT`. The following requirements apply:

- The aggregated expression must evaluate to a non-constant.
- The projection's `GROUP BY` clause must specify the aggregated expression.

For example, the following query uses `SUM(DISTINCT)` to calculate the total of all unique salaries in a given region:

```sql
SELECT customer_region, SUM(DISTINCT annual_income)::INT
FROM customer_dimension GROUP BY customer_region;
```
This query can use the following live aggregate projection, which specifies the aggregated column (annual_income) in its GROUP BY clause:

```
CREATE PROJECTION public.TotalRegionalIncome
(
  customer_region,
  annual_income,
  Count
)
AS
SELECT customer_dimension.customer_region,
       customer_dimension.annual_income,
       count(*) AS Count
FROM public.customer_dimension
GROUP BY customer_dimension.customer_region,
         customer_dimension.annual_income
;
```

**Note:** This projection includes the aggregate function COUNT, which here serves no logical objective; it is included only because live aggregate projections require at least one aggregate function.

### Creating Live Aggregate Projections

You define a live aggregate projection with the following syntax:

```
=> CREATE PROJECTION proj-name AS
    SELECT select-expression FROM table
    GROUP BY group-expression;
```

For full syntax options, see [CREATE PROJECTION (Live Aggregate Projections)](https://www.vertica.com/docs/9.0.x/VerticaAPIDocumentation.html).

For example:

```
=> CREATE PROJECTION clicks_agg AS
    SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
    GROUP BY page_id, click_time::DATE KSAFE 1;
```

For an extended discussion, see [Live Aggregate Projection Example](https://www.vertica.com/docs/9.0.x/VerticaAPIDocumentation.html).

### Requirements

The following requirements apply to live aggregate projections:

- **Live aggregate projections must be enabled:** configuration parameter EnableGroupByProjections must be set to 1 (default).
- The projection cannot be unsegmented.
SELECT and GROUP BY columns must be in the same order. GROUP BY expressions must be at the beginning of the SELECT list.

Restrictions

The following restrictions apply to live aggregate projections:

- Vertica does not regard the projections as superprojections, even one that includes all table columns.

- You cannot perform the following operations on anchor table data:
  
  DELETE
  UPDATE
  MERGE

  Note: To modify existing anchor table data, you must first drop all live aggregate and Top-K projections that are associated with it, using DROP PROJECTION.

- The projections can reference only one table.

Live Aggregate Projection Example

This example shows how you can track user clicks on a given web page using the following clicks table:

```sql
=> CREATE TABLE clicks(
    user_id INTEGER,
    page_id INTEGER,
    click_time TIMESTAMP NOT NULL);
```

You can aggregate user-specific activity with the following query:

```sql
=> SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
   WHERE click_time::DATE = '2015-04-30'
   GROUP BY page_id, click_time::DATE ORDER BY num_clicks DESC;
```

To facilitate performance of this query, create a live aggregate projection that counts the number of clicks per user:

```sql
=> CREATE PROJECTION clicks_agg AS
   SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
   GROUP BY page_id, click_time::DATE KSAFE 1;
```

When you query the clicks table on user clicks, Vertica typically directs the query to the live aggregate projection clicks_agg. As additional data is loaded into clicks, Vertica pre-
aggregates the new data and updates clicks_agg, so queries always return with the latest data.

For example:

```sql
=> SELECT page_id, click_time::DATE click_date, COUNT(*) num_clicks FROM clicks
  WHERE click_time::DATE = '2015-04-30' GROUP BY page_id, click_time::DATE
  ORDER BY num_clicks DESC;
```

<table>
<thead>
<tr>
<th>page_id</th>
<th>click_date</th>
<th>num_clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>2015-04-30</td>
<td>10</td>
</tr>
<tr>
<td>3003</td>
<td>2015-04-30</td>
<td>3</td>
</tr>
<tr>
<td>2003</td>
<td>2015-04-30</td>
<td>1</td>
</tr>
<tr>
<td>2035</td>
<td>2015-04-30</td>
<td>1</td>
</tr>
<tr>
<td>12034</td>
<td>2015-04-30</td>
<td>1</td>
</tr>
</tbody>
</table>

(5 rows)

Top-K Projections

A Top-K query returns the top \( k \) rows from partitions of selected rows. Top-K projections can significantly improve performance of Top-K queries. For example, you can define a table that stores gas meter readings with three columns: gas meter ID, time of meter reading, and the read value:

```sql
=> CREATE TABLE readings (meter_id INT,
                          reading_date TIMESTAMP,
                          reading_value FLOAT);
```

Given this table, the following Top-K query returns the five most recent meter readings for a given meter:

```sql
SELECT meter_id, reading_date, reading_value FROM readings
  LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

To improve the performance of this query, you can create a Top-K projection, which is a special type of live aggregate projection:

```sql
=> CREATE PROJECTION readings_topk (meter_id, recent_date, recent_value)
  AS SELECT meter_id, reading_date, reading_value FROM readings
  LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

After you create this Top-K projection and load its data (through `START_REFRESH` or `REFRESH`), Vertica typically redirects the query to the projection and returns with the pre-aggregated data.
Creating Top-K Projections

You define a Top-K projection with the following syntax:

```
CREATE PROJECTION proj-name [(proj-column-spec)]
    AS SELECT select-expression FROM table
    LIMIT num-rows OVER (PARTITION BY expression ORDER BY column-expr);
```

For full syntax options, see CREATE PROJECTION (Live Aggregate Projections).

For example:

```
=> CREATE PROJECTION readings_topk (meter_id, recent_date, recent_value)
    AS SELECT meter_id, reading_date, reading_value FROM readings
    LIMIT 5 OVER (PARTITION BY meter_id ORDER BY reading_date DESC);
```

For an extended discussion, see Top-K Projection Examples.

Requirements

The following requirements apply to Top-K projections:

- **Top-K projections must be enabled**: configuration parameter EnableTopKProjections must be set to 1 (default).
- The projection cannot be unsegmented.
- The window partition clause must use PARTITION BY.
- Columns in PARTITION BY and ORDER BY clauses must be the first columns specified in the SELECT list.
- You must use the LIMIT option to create a Top-K projection, instead of subqueries. For example, the following SELECT statements are equivalent:

```
=> SELECT symbol, trade_time last_trade, price last_price FROM (  
    SELECT symbol, trade_time, price, ROW_NUMBER()  
    OVER(PARTITION BY symbol ORDER BY trade_time DESC) rn FROM trades) trds WHERE rn <=1;
```

```
=> SELECT symbol, trade_time last_trade, price last_price FROM trades  
    LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

Both return the same results:
A Top-K projection that pre-aggregates data for use by both queries must include the LIMIT option:

```sql
=> CREATE PROJECTION trades_topk AS
   SELECT symbol, trade_time last_trade, price last_price FROM trades
   LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

Restrictions

The following restrictions apply to Top-K projections:

- Vertica does not regard the projections as superprojections, even one that includes all table columns.

- You cannot perform the following operations on anchor table data:
  - `DELETE`
  - `UPDATE`
  - `MERGE`

  **Note:** To modify existing anchor table data, you must first drop all live aggregate and Top-K projections that are associated with it, using `DROP PROJECTION`.

- The projections can reference only one table.

Top-K Projection Examples

The following examples show how to query a table with two Top-K projections for the most-recent trade and last trade of the day for each stock symbol.

1. Create a table that contains information about individual stock trades:
   - Stock symbol
   - Timestamp
   - Price per share
   - Number of shares
2. Load data into the table:

```sql
INSERT INTO trades VALUES('AAPL','2010-10-10 10:10:10'::TIMESTAMP,100.00,100);
INSERT INTO trades VALUES('AAPL','2010-10-10 10:10:10.3'::TIMESTAMP,101.00,100);
INSERT INTO trades VALUES ('AAPL','2011-10-10 10:10:10.5'::TIMESTAMP,106.1,1000);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.2'::TIMESTAMP,105.2,500);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.3'::TIMESTAMP,106.1,1000);
INSERT INTO trades VALUES ('HPQ','2012-10-10 10:10:10.4'::TIMESTAMP,42.05,100);
COMMIT;
```

3. Create two Top-K projections that obtain the following information from the `trades` table:

- Return the most recent trades for each stock symbol.
- Return the last trade on each trading day.

**For each stock symbol, return the most recent trade.**

```sql
=> CREATE PROJECTION trades_topk_a AS SELECT symbol, trade_time last_trade, price last_price
FROM trades LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);

=> SELECT symbol, trade_time last_trade, price last_price FROM trades
LIMIT 1 OVER(PARTITION BY symbol ORDER BY trade_time DESC);
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>last_trade</th>
<th>last_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPQ</td>
<td>2012-10-10 10:10:10.4</td>
<td>42.0500</td>
</tr>
<tr>
<td>AAPL</td>
<td>2011-10-10 10:10:10.5</td>
<td>106.1000</td>
</tr>
<tr>
<td>(2 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**For each stock symbol, return the last trade on each trading day.**

```sql
=> CREATE PROJECTION trades_topk_b
AS SELECT symbol, trade_time::DATE trade_date, trade_time, price close_price, volume
FROM trades LIMIT 1 OVER(PARTITION BY symbol, trade_time::DATE ORDER BY trade_time DESC);

=> SELECT symbol, trade_time::DATE trade_date, trade_time, price close_price, volume
FROM trades LIMIT 1 OVER(PARTITION BY symbol, trade_time::DATE ORDER BY trade_time DESC);
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>trade_date</th>
<th>trade_time</th>
<th>close_price</th>
<th>volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2 rows)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In each scenario, Vertica redirects queries on the trades table to the appropriate Top-K projection and returns the aggregated data from them. As additional data is loaded into this table, Vertica pre-aggregates the new data and updates the Top-K projections, so queries always return with the latest data.

Pre-Aggregating UDTF Results

CREATE PROJECTION can define live aggregate projections that invoke user-defined transform functions (UDTFs). To minimize overhead when you query those projections, Vertica processes these functions in the background and stores their results on disk.

Important: Currently, live aggregate projections can only reference UDTFs that are developed in C++.

Defining Projections with UDTFs

The projection definition characterizes UDTFs in one of two ways:

- Identifies the UDTF as a pre-pass UDTF, which transforms newly loaded data before it is stored in the projection ROS containers.
- Identifies the UDTF as a batch UDTF, which aggregates and stores projection data.

The projection definition identifies a UDTF as a pre-pass UDTF or batch UDTF in its window partition clause, through the keywords PREPASS or BATCH. A projection can specify one pre-pass or batch UDTF or include both (see UDTF Specification Options).

In all cases, the projection is implicitly segmented and ordered on the PARTITION BY columns.

UDTF Specification Options

Projections can invoke batch and pre-pass UDTFs singly or in combination.
Single Pre-Pass UDTF

Vertica invokes the pre-pass UDTF when you load data into the projection's anchor table—for example through COPY or INSERT statements. A pre-pass UDTF transforms the new data and then stores the transformed data in the projection's ROS containers.

Use the following syntax:

```sql
=> CREATE PROJECTION projection-name AS SELECT ..., udf(args)
    OVER(PARTITION PREPASS BY partition-col-exprs) FROM table-ref;
```

Single Batch UDTF

When invoked singly, a batch UDTF transforms and aggregates projection data on mergeout, data load, and query operations. The UDTF stores aggregated results in the projection's ROS containers. Aggregation is cumulative across mergeout and load operations, and is completed (if necessary) on query execution.

Use the following syntax:

```sql
=> CREATE PROJECTION projection-name AS SELECT ..., udf(args)
    OVER(PARTITION BATCH BY partition-cols) AS (batch-output-columns) FROM table-ref;
```

Combined Pre-Pass and Batch UDTFs

You can define a projection with a subquery that invokes a pre-pass UDTF. The pre-pass UDTF returns transformed data to the outer batch query. The batch UDTF then iteratively aggregates results across mergeout operations. It completes aggregation (if necessary) on query execution.

Use the following syntax:

```sql
=> CREATE PROJECTION projection-name AS SELECT ..., batch-udtf(batch-args)
    OVER ( PARTITION BATCH BY partition-cols ) AS (batch-output-columns)
    FROM ( SELECT ..., prepass-udtf(prepass-args)
        OVER ( PARTITION PREPASS BY partition-cols ) AS (prepass-output-columns)
        FROM table-ref ) sq-ref;
```

Important: The outer batch UDTF arguments `batch-args` must exactly match the output columns returned by the pre-pass UDTF, in name and order.
Examples

Single pre-pass UDTF

The following example shows how to use the UDTF `text_index`, which extracts from a text document strings that occur more than once.

The following projection specifies to invoke `text_index` as a pre-pass UDTF:

```sql
=> CREATE TABLE documents ( doc_id INT PRIMARY KEY, text VARCHAR(140));
=> CREATE PROJECTION index_proj
   AS SELECT doc_id, text_index(doc_id, text)
   OVER (PARTITION PREPASS BY doc_id) FROM documents;
```

The UDTF is invoked whenever data is loaded into the anchor table `documents`. `text_index` transforms the newly loaded data, and Vertica stores the transformed data in the live aggregate projection ROS containers.

So, if you load the following data into `documents`:

```sql
=> INSERT INTO documents VALUES (100, 'A SQL Query walks into a bar. In one corner of the bar are two tables.
OUTPUT
-------
1
(1 row)
```

text_index transforms the newly loaded data and stores it in the projection ROS containers. When you query the projection, it returns with the following results:

<table>
<thead>
<tr>
<th>doc_id</th>
<th>frequency</th>
<th>term</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
<td>bar</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>Query</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>tables</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>the</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>walks</td>
</tr>
</tbody>
</table>

Combined Pre-Pass and Batch UDTFs

The following projection specifies pre-pass and batch UDTFs `stv_intersect` and `aggregate_classified_points`, respectively:

```sql
CREATE TABLE points( point_id INTEGER, point_type VARCHAR(10), coordinates GEOMETRY(100));
CREATE PROJECTION aggregated_proj
   AS SELECT point_type, aggregate_classified_points( sq.point_id, sq.polygon_id)
   OVER (PARTITION BATCH BY point_type)
   FROM
   (SELECT point_type, stv_intersect( points)

```
point_id, coordinates USING PARAMETERS index='polygons') 
OVER (PARTITION PREPASS BY point_type) AS (point_id, polygon_id) FROM points) sq;

The pre-pass query UDTF stv_intersect returns its results (a set of point and matching polygon IDs) to the outer batch query. The outer batch query then invokes the UDTF aggregate_classified_points. Vertica aggregates the result set that is returned by aggregate_classified_points whenever a mergeout operation consolidates projection data. Final aggregation (if necessary) occurs when the projection is queried.

The batch UDTF arguments must exactly match the output columns returned by the pre-pass UDTF stv_intersect, in name and order. In this example, the pre-pass subquery explicitly names the pre-pass UDTF output columns point_id and polygon_id. Accordingly, the batch UDTF arguments match them in name and order: sq.point_id and sq.polygon_id.

### Aggregating Data Through Expressions

You can create projections where one or more columns are defined by expressions. An expression can reference one or more anchor table columns. For example, the following table contains two integer columns, a and b:

```
=> CREATE TABLE values (a INT, b INT);
```

You can create a projection with an expression that calculates the value of column c as the product of a and b:

```
=> CREATE PROJECTION values_product (a, b, c) 
   AS SELECT a, b, a*b FROM values SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

When you load data into this projection, Vertica resolves the expression a*b in column c. You can then query the projection instead of the anchor table. Vertica returns the pre-calculated data and avoids the overhead otherwise incurred by resource-intensive computations.

Using expressions in projections also lets you sort or segment data on the calculated results of an expression instead of sorting on single column values.

**Note:** If a projection with expressions also includes aggregate functions such as `SUM` or `COUNT`, Vertica treats it like a live aggregate projection.

### Support for User-Defined Scalar Functions

**Important:** Currently, support for pre-aggregating UDSF results is limited to C++.
Vertica treats user-defined scalar functions (UDSFs) like other expressions. On each load operation, the UDSF is invoked and returns its results. Vertica stores these results on disk, and returns them when you query the projection directly.

In the following example, the projection points_p1 specifies the UDSF zorder, which is invoked whenever data is loaded in the anchor table points. When data is loaded into the projection, Vertica invokes this function and stores its results for fast access by future queries.

```sql
=> CREATE TABLE points(point_id INTEGER, lat NUMERIC(12,9), long NUMERIC(12,9));
=> CREATE PROJECTION points_p1
   AS SELECT point_id, lat, long, zorder(lat, long) zorder
          FROM points
   ORDER BY zorder(lat, long) SEGMENTED BY hash(point_id) ALL NODES;
```

**Requirements**

- **Projections with expressions must be enabled.**
- Any `ORDER BY` expression must be in the `SELECT` list.
- All projection columns must be named.

**Restrictions**

- You can delete and update data in the anchor table, but you cannot perform a merge operation.
- Unlike live aggregate projections, Vertica does not redirect queries with expressions to an equivalent existing projection.
- Projection expressions must be immutable—that is, they must always return the same result. For example, a projection cannot include expressions that use `TO CHAR` (depends on locale) or `RANDOM` (returns different value at each invocation).
- Projection expressions cannot include Vertica meta-functions such as `ADVANCE_EPOCH`, `ANALYZE_STATISTICS`, `EXPORT_TABLES`, or `START_REFRESH`.

**Querying Data Through Expressions Example**

The following example uses a table that contains two integer columns, a and b:

```sql
=> CREATE TABLE values (a INT, b INT);
```
You can create a projection with an expression that calculates the value of column c as the product of a and b:

```sql
=> CREATE PROJECTION values_product (a, b, c)
   AS SELECT a, b, a*b FROM values SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

To query this projection, use the name that Vertica assigns to it or to its buddy projections. For example, the following queries target different instances of the projection defined earlier, and return the same results:

```sql
=> SELECT * FROM values_product_b0;
=> SELECT * FROM values_product_b1;
```

The following example queries the anchor table:

```sql
=> SELECT * FROM values;
```

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>111</td>
</tr>
</tbody>
</table>
```

Given the projection created earlier, querying that projection returns the following values:

```
VMart=> SELECT * FROM values_product_b0;
```

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>165</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>72</td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>184</td>
</tr>
<tr>
<td>16</td>
<td>41</td>
<td>656</td>
</tr>
<tr>
<td>22</td>
<td>111</td>
<td>2442</td>
</tr>
</tbody>
</table>
```

**System Table Fields**

You can query the following system table fields for information about live aggregate projections, Top-K projections, and projections with expressions:

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLES</td>
<td>HASAggregate_PROJECTION</td>
</tr>
<tr>
<td>Table</td>
<td>Fields</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>PROJECTIONS</td>
<td>AGGREGATE_TYPE</td>
</tr>
<tr>
<td></td>
<td>HAS_EXPRESSIONS</td>
</tr>
<tr>
<td></td>
<td>AGGREGATE_TYPE</td>
</tr>
<tr>
<td>PROJECTION_COLUMNS</td>
<td>COLUMN_EXPRESSION</td>
</tr>
<tr>
<td></td>
<td>IS_AGGREGATE</td>
</tr>
<tr>
<td></td>
<td>IS_EXPRESSION</td>
</tr>
<tr>
<td></td>
<td>ORDER_BY_POSITION</td>
</tr>
<tr>
<td></td>
<td>ORDER_BY_TYPE</td>
</tr>
<tr>
<td></td>
<td>PARTITION_BY_POSITION</td>
</tr>
</tbody>
</table>
Event Series Joins

An event series join is a Vertica SQL extension that enables the analysis of two series when their measurement intervals don’t align precisely, such as with mismatched timestamps. You can compare values from the two series directly, rather than having to normalize the series to the same measurement interval.

Event series joins are an extension of Outer Joins, but instead of padding the non-preserved side with NULL values when there is no match, the event series join pads the non-preserved side values that it interpolates from the previous value.

The difference in how you write a regular join versus an event series join is the INTERPOLATE predicate, which is used in the ON clause. For example, the following two statements show the differences, which are shown in greater detail in Writing Event Series Joins.

<table>
<thead>
<tr>
<th>Regular full outer join</th>
<th>Event series join</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM hTicks h FULL OUTER JOIN aTicks a</td>
<td>SELECT * FROM hTicks h FULL OUTER JOIN aTicks a</td>
</tr>
<tr>
<td>ON (h.time = a.time);</td>
<td>ON (h.time INTERPOLATE PREVIOUS VALUE a.time);</td>
</tr>
</tbody>
</table>

Similar to regular joins, an event series join has inner and outer join modes, which are described in the topics that follow.

For full syntax, including notes and restrictions, see INTERPOLATE in the SQL Reference Manual

Sample Schema for Event Series Joins

Examples

If you don't plan to run the queries and just want to look at the examples, you can skip this topic and move straight to Writing Event Series Joins.

Schema of hTicks and aTicks Tables

The examples that follow use the following hTicks and aTicks tables schemas:

```sql
CREATE TABLE hTicks (  
    stock VARCHAR(20),  
    time TIME,  
    price NUMERIC(8,2)  
);  
CREATE TABLE aTicks (  
```
Although TIMESTAMP is more commonly used for the event series column, the examples in this topic use TIME to keep the output simple.

```sql
INSERT INTO hTicks VALUES ('HPQ', '12:00', 50.00);
INSERT INTO hTicks VALUES ('HPQ', '12:01', 51.00);
INSERT INTO hTicks VALUES ('HPQ', '12:05', 51.00);
INSERT INTO hTicks VALUES ('HPQ', '12:06', 52.00);
INSERT INTO aTicks VALUES ('ACME', '12:00', 340.00);
INSERT INTO aTicks VALUES ('ACME', '12:03', 340.10);
INSERT INTO aTicks VALUES ('ACME', '12:05', 340.20);
INSERT INTO aTicks VALUES ('ACME', '12:05', 333.80);
COMMIT;
```

Output of the two tables:

<table>
<thead>
<tr>
<th>stock</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPQ</td>
<td>12:00:00</td>
<td>50.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:01:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:05:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:06:00</td>
<td>52.00</td>
</tr>
</tbody>
</table>

(4 rows)

<table>
<thead>
<tr>
<th>stock</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>12:00:00</td>
<td>340.00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:03:00</td>
<td>340.10</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>340.20</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>333.80</td>
</tr>
</tbody>
</table>

(4 rows)

**Example Query Showing Gaps**

A full outer join shows the gaps in the timestamps:

```sql
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON h.time = a.time;
stock | time   | price | stock | time   | price |
-------|--------|-------|-------|--------|-------|
HPQ    | 12:00:00 | 50.00 | ACME  | 12:00:00 | 340.00 |
HPQ    | 12:01:00 | 51.00 |       |        |       |
HPQ    | 12:05:00 | 51.00 | ACME  | 12:05:00 | 333.80 |
HPQ    | 12:05:00 | 51.00 | ACME  | 12:05:00 | 340.20 |
HPQ    | 12:06:00 | 52.00 |       |        |       |
|       |        |       | ACME  | 12:03:00 | 340.10 |
(6 rows)
```
Schema of Bid and Asks Tables

The examples that follow use the following hTicks and aTicks tables.

```sql
CREATE TABLE bid(stock VARCHAR(20), time TIME, price NUMERIC(8,2));
CREATE TABLE ask(stock VARCHAR(20), time TIME, price NUMERIC(8,2));
INSERT INTO bid VALUES ('HPQ', '12:00', 100.10);
INSERT INTO bid VALUES ('HPQ', '12:01', 100.00);
INSERT INTO bid VALUES ('ACME', '12:00', 80.00);
INSERT INTO bid VALUES ('ACME', '12:03', 79.80);
INSERT INTO bid VALUES ('ACME', '12:05', 79.90);
INSERT INTO ask VALUES ('HPQ', '12:01', 101.00);
INSERT INTO ask VALUES ('ACME', '12:00', 80.00);
INSERT INTO ask VALUES ('ACME', '12:02', 75.00);
COMMIT;
```

Output of the two tables:

<table>
<thead>
<tr>
<th>bid</th>
<th>ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>=&gt; SELECT * FROM bid;</td>
<td>=&gt; SELECT * FROM ask;</td>
</tr>
<tr>
<td>There are no entry records for stocks HPQ and ACME at 12:02 and at 12:04:</td>
<td>There are no entry records for stock HPQ at 12:00 and none for ACME at 12:01:</td>
</tr>
<tr>
<td>stock</td>
<td>time</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:00:00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:01:00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:00:00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:03:00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
</tr>
<tr>
<td>(5 rows)</td>
<td></td>
</tr>
</tbody>
</table>

Example Query Showing Gaps

A full outer join shows the gaps in the timestamps:

```sql
=> SELECT * FROM bid b FULL OUTER JOIN ask a ON b.time = a.time;
stock | time | price | stock | time | price
---------------------------------------------
HPQ   | 12:00:00 | 100.10 | ACME  | 12:00:00 | 80.00
HPQ   | 12:01:00 | 100.00 | HPQ   | 12:01:00 | 101.00
ACME  | 12:00:00 | 80.00  | ACME  | 12:00:00 | 80.00
ACME  | 12:03:00 | 79.80  | ACME  | 12:02:00 | 75.00
ACME  | 12:05:00 | 79.90  |
(6 rows)
```
**Writing Event Series Joins**

The examples in this topic contain mismatches between timestamps—just as you’d find in real life situations; for example, there could be a period of inactivity on stocks where no trade occurs, which can present challenges when you want to compare two stocks whose timestamps don’t match.

**The hTicks and aTicks Tables**

As described in the example ticks schema, tables, hTicks is missing input rows for 12:02, 12:03, and 12:04, and aTicks is missing inputs at 12:01, 12:02, and 12:04.

```
<table>
<thead>
<tr>
<th>stock</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPQ</td>
<td>12:00:00</td>
<td>50.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:01:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:05:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:06:00</td>
<td>52.00</td>
</tr>
</tbody>
</table>
```

(4 rows)

```
<table>
<thead>
<tr>
<th>stock</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>12:00:00</td>
<td>340.00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:03:00</td>
<td>340.10</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>340.20</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>333.80</td>
</tr>
</tbody>
</table>
```

(4 rows)

**Querying Event Series Data with Full Outer Joins**

Using a traditional full outer join, this query finds a match between tables hTicks and aTicks at 12:00 and 12:05 and pads the missing data points with NULL values.

```
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a ON (h.time = a.time);
<table>
<thead>
<tr>
<th>stock</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPQ</td>
<td>12:00:00</td>
<td>50.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:01:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:05:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:05:00</td>
<td>51.00</td>
</tr>
<tr>
<td>HPQ</td>
<td>12:06:00</td>
<td>52.00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:00:00</td>
<td>340.00</td>
</tr>
<tr>
<td>ACME</td>
<td>12:01:00</td>
<td>340.10</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>340.20</td>
</tr>
<tr>
<td>ACME</td>
<td>12:05:00</td>
<td>333.80</td>
</tr>
</tbody>
</table>
```

(6 rows)

To replace the gaps with interpolated values for those missing data points, use the INTERPOLATE predicate to create an event series join. The join condition is restricted to the ON clause, which evaluates the equality predicate on the timestamp columns from the two input
In other words, for each row in outer table hTicks, the ON clause predicates are evaluated for each combination of each row in the inner table aTicks.

Simply rewrite the full outer join query to use the INTERPOLATE predicate with the required PREVIOUS VALUE keywords. Note that a full outer join on event series data is the most common scenario for event series data, where you keep all rows from both tables.

```sql
=> SELECT * FROM hTicks h FULL OUTER JOIN aTicks a
  ON (h.time INTERPOLATE PREVIOUS VALUE a.time);
```

Vertica interpolates the missing values (which appear as NULL in the full outer join) using that table’s previous value:

```
stock | time  | price | stock | time  | price
-------+--------+--------+--------+--------+--------
HPQ    | 12:00:00 | 50.00  | ACME   | 12:00:00 | 340.00
HPQ    | 12:01:00 | 51.00  | ACME   | 12:00:00 | 340.00
HPQ    | 12:01:00 | 51.00  | ACME   | 12:03:00 | 340.10
HPQ    | 12:05:00 | 51.00  | ACME   | 12:05:00 | 333.80
HPQ    | 12:05:00 | 51.00  | ACME   | 12:05:00 | 340.20
HPQ    | 12:06:00 | 52.00  | ACME   | 12:05:00 | 340.20
(6 rows)
```

**Note:** The output ordering above is different from the regular full outer join because in the event series join, interpolation occurs independently for each stock (hTicks and aTicks), where the data is partitioned and sorted based on the equality predicate. This means that interpolation occurs within, not across, partitions.

If you review the regular full outer join output, you can see that both tables have a match in the time column at 12:00 and 12:05, but at 12:01, there is no entry record for ACME. So the operation interpolates a value for ACME (ACME, 12:00, 340) based on the previous value in the aTicks table.

### Querying Event Series Data with Left Outer Joins

You can also use left and right outer joins. You might, for example, decide you want to preserve only hTicks values. So you’d write a left outer join:

```sql
=> SELECT * FROM hTicks h LEFT OUTER JOIN aTicks a
  ON (h.time INTERPOLATE PREVIOUS VALUE a.time);
```

```
stock | time  | price | stock | time  | price
-------+--------+--------+--------+--------+--------
HPQ    | 12:00:00 | 50.00  | ACME   | 12:00:00 | 340.00
```
Here's what the same data looks like using a traditional left outer join:

```sql
=> SELECT * FROM hTicks h LEFT OUTER JOIN aTicks a ON h.time = a.time;
stock | time | price | stock | time | price
------------+-------+-------+--------+-------+-------
HPQ  | 12:00:00 | 50.00 | ACME   | 12:00:00 | 340.00
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 333.80
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 340.20
HPQ  | 12:06:00 | 52.00 | ACME   | 12:05:00 | 340.20
(5 rows)
```

Note that a right outer join has the same behavior with the preserved table reversed.

### Querying Event Series Data with Inner Joins

Note that INNER event series joins behave the same way as normal ANSI SQL-99 joins, where all gaps are omitted. Thus, there is nothing to interpolate, and the following two queries are equivalent and return the same result set:

**A regular inner join:**

```sql
=> SELECT * FROM hTicks h JOIN aTicks a
   ON (h.time INTERPOLATE PREVIOUS VALUE a.time);
stock | time | price | stock | time | price
------------+-------+-------+--------+-------+-------
HPQ  | 12:00:00 | 50.00 | ACME   | 12:00:00 | 340.00
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 333.80
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 340.20
(3 rows)
```

**An event series inner join:**

```sql
=> SELECT * FROM hTicks h INNER JOIN aTicks a ON (h.time = a.time);
stock | time | price | stock | time | price
------------+-------+-------+--------+-------+-------
HPQ  | 12:00:00 | 50.00 | ACME   | 12:00:00 | 340.00
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 333.80
HPQ  | 12:05:00 | 51.00 | ACME   | 12:05:00 | 340.20
(3 rows)
```
The Bid and Ask Tables

Using the example schema for the bid and ask tables, write a full outer join to interpolate the missing data points:

```sql
=> SELECT * FROM bid b FULL OUTER JOIN ask a
   ON (b.stock = a.stock AND b.time INTERPOLATE PREVIOUS VALUE a.time);
```

In the below output, the first row for stock HPQ shows nulls because there is no entry record for HPQ before 12:01.

```
stock | time | price | stock | time | price
-------+-------+-------+-------+-------+-------
ACME   | 12:00:00 | 80.00 | ACME   | 12:00:00 | 80.00 
ACME   | 12:00:00 | 80.00 | ACME   | 12:02:00 | 75.00 
ACME   | 12:03:00 | 79.80 | ACME   | 12:02:00 | 75.00 
ACME   | 12:05:00 | 79.90 | ACME   | 12:02:00 | 75.00 
HPQ    | 12:00:00 | 100.10 |         |       |       
HPQ    | 12:01:00 | 100.00 | HPQ    | 12:01:00 | 101.00 
```

(6 rows)

Note also that the same row (ACME, 12:02, 75) from the ask table appears three times. The first appearance is because no matching rows are present in the bid table for the row in ask, so Vertica interpolates the missing value using the ACME value at 12:02 (75.00). The second appearance occurs because the row in bid (ACME, 12:05, 79.9) has no matches in ask. The row from ask that contains (ACME, 12:02, 75) is the closest row; thus, it is used to interpolate the values.

If you write a regular full outer join, you can see where the mismatched timestamps occur:

```sql
=> SELECT * FROM bid b FULL OUTER JOIN ask a ON (b.time = a.time);
```

```
stock | time | price | stock | time | price
-------+-------+-------+-------+-------+-------
ACME   | 12:00:00 | 80.00 | ACME   | 12:00:00 | 80.00 
ACME   | 12:03:00 | 79.80 | ACME   | 12:02:00 | 75.00 
ACME   | 12:05:00 | 79.90 | ACME   | 12:02:00 | 75.00 
HPQ    | 12:00:00 | 100.10 |         |       |       
HPQ    | 12:01:00 | 100.00 | HPQ    | 12:01:00 | 101.00 
```

(6 rows)
Event Series Pattern Matching

The SQL MATCH Clause syntax (described in the SQL Reference Manual) lets you screen large amounts of historical data in search of event patterns. You specify a pattern as a regular expression and can then search for the pattern within a sequence of input events. MATCH provides subclauses for analytic data partitioning and ordering, and the pattern matching occurs on a contiguous set of rows.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- landed on the company home page
- navigated to the product page
- ran a search
- clicked a link from the search results
- made a purchase

Clickstream Funnel Schema

The examples in this topic use this clickstream funnel and the following clickstream_log table schema:

```sql
=> CREATE TABLE clickstream_log (  
  uid INT, --user ID  
  sid INT, --browsing session ID, produced by previous sessionization computation  
  ts TIME, --timestamp that occurred during the user's page visit  
  refURL VARCHAR(20), --URL of the page referencing PageURL  
  pageURL VARCHAR(20), --URL of the page being visited  
  action CHAR(1) --action the user took after visiting the page ('P' = Purchase, 'V' = View)  
);
```

```sql
INSERT INTO clickstream_log VALUES (1,100,'12:00','website1.com','website2.com/home', 'V');
INSERT INTO clickstream_log VALUES (1,100,'12:01','website2.com/home','website2.com/floby', 'V');
INSERT INTO clickstream_log VALUES (1,100,'12:02','website2.com/floby','website2.com/shamwow', 'V');
INSERT INTO clickstream_log VALUES (1,100,'12:03','website2.com/shamwow','website2.com/buy', 'P');
```
INSERT INTO clickstream_log values (2,100,'12:10','website1.com','website2.com/home', 'V');
INSERT INTO clickstream_log values (2,100,'12:11','website2.com/home','website2.com/forks', 'V');
INSERT INTO clickstream_log values (2,100,'12:13','website2.com/forks','website2.com/buy', 'P');
COMMIT;

Here's the clickstream_log table's output:

<table>
<thead>
<tr>
<th>uid</th>
<th>sid</th>
<th>ts</th>
<th>refURL</th>
<th>pageURL</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>12:00:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:01:00</td>
<td>website2.com/home</td>
<td>website2.com/floby</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:02:00</td>
<td>website2.com/floby</td>
<td>website2.com/shamwow</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:03:00</td>
<td>website2.com/shamwow</td>
<td>website2.com/buy</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:10:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:11:00</td>
<td>website2.com/home</td>
<td>website2.com/forks</td>
<td>V</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:13:00</td>
<td>website2.com/forks</td>
<td>website2.com/buy</td>
<td>P</td>
</tr>
</tbody>
</table>

(7 rows)

Example

This example includes the Vertica Pattern Matching Functions to analyze users' browsing history over website2.com. It identifies patterns where the user performed the following tasks:

- Landed on website2.com from another web site (Entry)
- Browsed to any number of other pages (Onsite)
- Made a purchase (Purchase)

In the following statement, pattern P (Entry Onsite* Purchase) consist of three event types: Entry, Onsite, and Purchase. When Vertica finds a match in the input table, the associated pattern instance must be an event of type Entry followed by 0 or more events of type Onsite, and an event of type Purchase

```sql
=> SELECT uid, sid, ts, refurl, pageurl, action, event_name(), pattern_id(), match_id() FROM clickstream_log MATCH (PARTITION BY uid, sid ORDER BY ts DEFINE Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
```

Vertica Documentation
Analyzing Data
In the output below, the first four rows represent the pattern for user 1's browsing activity, while the following three rows show user 2's browsing habits.

```
uid | sid | ts   | refurl    | pageurl         | action | event_name    | pattern_id | match_id |
---  | ---  | ---   | --------- | --------------- | ------ | --------  | --------- | ------- |
1    | 100  | 12:00 | website1.com | website2.com/home | V     | Entry    | 1         |         |
1    | 100  | 12:01 | website2.com/home | website2.com/floby | V     | Onsite   | 2         |         |
1    | 100  | 12:02 | website2.com/floby | website2.com/shamwow | V     | Onsite   | 3         |         |
1    | 100  | 12:03 | website2.com/shamwow | website2.com/buy   | P     | Purchase | 4         |         |
2    | 100  | 12:10 | website1.com | website2.com/home | V     | Entry    | 1         |         |
1    | 100  | 12:11 | website2.com/home | website2.com/forks | V     | Onsite   | 2         |         |
2    | 100  | 12:13 | website2.com/forks | website2.com/buy   | P     | Purchase | 3         |         |
```

See Also

- MATCH Clause
- Pattern Matching Functions
Using Flex Tables

This guide describes how to use flexible (flex) tables, which are a different kind of database table designed for loading and querying unstructured data, also called *semi-structured* data in your Vertica Analytics Platform. Flex tables can contain only unstructured, raw data, or both unstructured and columnar data. You can create a flex table with or without a schema or real columns. Hybrid tables consist of both unstructured and real columns. Both flex and hybrid tables are fully supported Vertica Analytics Platform tables, stored as projections and with the same K-safety as your database.
Audience

This guide is intended for use by any user or database designer or application developer interested in working with flexible tables in the database.
Prerequisites

This guide assumes that Vertica Analytics Platform Version 9.0.x is installed and running in your environment.

It also assumes that you are familiar with using the Vertica Analytics Platform, especially the following features and commands:

- Loading data with the `COPY` statement and its basic parameters
- Using statements to create tables `CREATE TABLE` or CTAS (create table as...)
- Altering table definitions with the `ALTER TABLE` statement
- Creating and using Views (`CREATE VIEW`)
- Querying data using the `SELECT` statement
- Using functions for your database

If you are not familiar with these tasks and features, see Getting Started.
Getting Started

Getting Started describes the basics of creating, exploring, and using flex tables. The rest of this guide presents beyond the basics details using simple examples.

Create a Simple JSON File

Use this JSON data for the exercises in the rest of this section:

```json
{"name": "Everest", "type":"mountain", "height":29029, "hike_safety": 34.1}
{"name": "Mt St Helens", "type":"volcano", "height":29029, "hike_safety": 15.4}
{"name": "Denali", "type":"mountain", "height":17000, "hike_safety": 12.2}
{"name": "Kilimanjaro", "type":"mountain", "height":14000 }
{"name": "Mt Washington", "type":"mountain", "hike_safety": 50.6}
```

1. Copy and paste the JSON data into your favorite editor.
2. Save the file in any convenient location for loading into your Vertica database.

Create a Flex Table and Load Data

1. Create a flex table called mountains:

```sql
=> CREATE flex table mountains();
```

2. Load the JSON file you saved, using the flex table parser `fjsonparser`:

```sql
=> COPY mountains from '/home/dbadmin/data/flex/mountains.json' parser fjsonparser();
Rows Loaded
------------
5
(1 row)
```

3. Query values from the sample file:

```sql
=> SELECT name, type, height from mountains;
  name | type   | height
-------|--------|--------
 Everest | mountain | 29029
```
Vertica Documentation
Using Flex Tables

Mt St Helens
Denali
Kilimanjaro
Mt Washington
(5 rows)

|
|
|
|

volcano
mountain
mountain
mountain

| 29029
| 17000
| 14000
|

You have now created a flex table and loaded data. Next, learn more about using flex table
data in your database.

Query More of Your Flex Table
1. Query your flex table to see the data you loaded as it is stored in the __raw__ column.
The example illustrates the table contents, with Return characters added for illustration:
=> \x
Expanded display is on.
=> SELECT * from mountains;
[ RECORD 1 ]+---------------------------------------------------------------------------__identity__ | 1
__raw__
| \001\000\000\000,\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000$\000\000\0002902934.1Everestmountain\004\000\000\000\024\000\000\000\032\000\
000\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 2 ]+---------------------------------------------------------------------------__identity__ | 2
__raw__
| \001\000\000\0000\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000)\000\000\0002902915.4Mt St
Helensvolcano\004\000\000\000\024\000\000\000\032\000\
000\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 3 ]+---------------------------------------------------------------------------__identity__ | 3
__raw__
| \001\000\000\000+\000\000\000\004\000\000\000\024\000\000\000\031\000\000\000\
035\000\000\000#\000\000\0001700012.2Denalimountain\004\000\000\000\024\000\000\000\032\000\000
\000%\000\000\000)\000\000\000heighthike_safetynametype
[ RECORD 4 ]+---------------------------------------------------------------------------__identity__ | 4
__raw__
| \001\000\000\000(\000\000\000\003\000\000\000\020\000\000\000\025\000\000\000\
000\000\00014000Kilimanjaromountain\003\000\000\000\020\000\000\000\026\000\000\000\032\000\
000\000heightnametype
[ RECORD 5 ]+---------------------------------------------------------------------------__identity__ | 5
__raw__
| \001\000\000\000)\000\000\000\003\000\000\000\020\000\000\000\024\000\000\000\
000\000\00050.6Mt Washingtonmountain\003\000\000\000\020\000\000\000\033\000\000\000\037\000\
000\000hike_safetynametype

2. Use the mapToString() function (with the __raw__ column of mountains) to inspect
its contents in readable JSON text format:

Vertica Analytic Database (9.0.x)

Page 1733 of 6180


3. Now, use the `compute_flextable_keys()` function to populate the `mountain_keys` table. Vertica generates this table automatically when you create your flex table.

```sql
=> SELECT compute_flextable_keys('mountains');
compute_flextable_keys
-------------------------------------------------------------------------------
Please see public.mountains_keys for updated keys
(1 row)
```

4. Query the keys table (`mountains_keys`), and examine the results:

```sql
=> SELECT * from public.mountains_keys;
key_name | frequency | data_type_guess
---------+-----------+-------------------
  hike_safety |     4 | varchar(20)  
     name |     5 | varchar(26)  
    height |     4 | varchar(20)  
```
Build a Flex Table View

1. Use the `build_flextable_view()` function to populate a view generated from the `mountains_keys` table.

```sql
=> SELECT build_flextable_view('mountains');
build_flextable_view
--------------------
The view public.mountains_view is ready for querying
(1 row)
```

2. Query the view `mountains_view`:

```sql
=> SELECT * from public.mountains_view;
hike_safety | name         | type   | height
-------------|--------------|--------|--------
50.6         | Mt Washington| mountain|        
34.1         | Everest      | mountain| 29029  
22.8         | Kilimanjaro  | mountain| 14000  
15.4         | Mt St Helens | volcano | 29029  
12.2         | Denali       | mountain| 17000  
(5 rows)
```

3. Use the `view_columns` system table to query the `column_name` and `data_type` columns for `mountains_view`:

```sql
=> SELECT column_name, data_type from view_columns where table_name = 'mountains_view';
column_name | data_type
-------------|--------
hike_safety | varchar(20)
name         | varchar(20)
type         | varchar(20)
height       | varchar(20)
(4 rows)
```

4. Review the query results:

- Notice the `data_type` column, its values and sizes. These are calculated when you compute keys for your flex table with `compute_flextable_keys()`.

---

Vertica Analytic Database (9.0.x)  Page 1735 of 6180
Did you also notice the data_type_guess column when you queried the mountains_keys table after invoking that function?

5. With the data_type information from mountains_view, override the data_type_guess for hike_safety. Then, COMMIT the change, and rebuild the view with build_flextable_view():

```sql
=> UPDATE mountains_keys SET data_type_guess = 'float' where key_name = 'hike_safety';
OUTPUT
-------
  1
(1 row)
=> commit;
=> SELECT build_flextable_view('mountains');

build_flextable_view
----------------------
The view public.mountains_view is ready for querying
(1 row)
```

6. Next, use the view_columns system table. Notice that hike_safety is now a float data type:

```sql
=> SELECT column_name, data_type from view_columns where table_name = 'mountains_view';
column_name | data_type
-------------+----------
hike_safety  | float
name         | varchar(26)
type         | varchar(20)
height       | varchar(20)
(4 rows)
```

### Create a Hybrid Flex Table

If you already know that some of the data you load and query regularly needs full Vertica performance and support, you can create a hybrid flex table. A hybrid flex table has one or more real columns that you define, and a __raw__ column to store any unstructured data you load. Querying real columns is faster than querying flexible data in the __raw__ column. You can define default values for the columns.

1. Create a hybrid flex table, and load the same sample JSON file:

```sql
=> CREATE flex table mountains_hybrid(name varchar(41) default name::varchar(41), hike_safety float
```
2. Use the `compute_flextable_keys_and_build_view()` function to populate the keys table and build the view for `mountains_hybrid`:

```sql
=> SELECT compute_flextable_keys_and_build_view('mountains_hybrid');
compute_flextable_keys_and_build_view

Please see public.mountains_hybrid_keys for updated keys
The view public.mountains_hybrid_view is ready for querying
(1 row)
```

3. Query the `mountains_hybrid_keys` table. Review the `data_type_guesses` column values again. The types list the column definitions you declared when you created the hybrid table:

```sql
=> SELECT * from mountains_hybrid_keys;
key_name | frequency | data_type_guess
---------|-----------|------------------
height   | 4         | varchar(20)
name     | 5         | varchar(41)
type     | 5         | varchar(20)
hike_safety | 4    | float

(4 rows)
```

If you create a basic flex table, and later find you want to promote one or more virtual columns to real columns, see Materializing Flex Tables to add columns.

**Materialize Virtual Columns in a Hybrid Flex Table**

After you explore your flex table data, you can promote one or more virtual columns in your flex table to real columns. You do not need to create a separate columnar table.
1. **Invoke the `materialize_flextable_columns()` function on the hybrid table, specifying the number of virtual columns to materialize:**

```sql
=> SELECT materialize_flextable_columns('mountains_hybrid', 3);

materialize_flextable_columns

The following columns were added to the table public.mountains_hybrid:

type

For more details, run the following query:

```sql
SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and table_name = 'mountains_hybrid';
```

(1 row)

2. You specified three (3) columns to materialize, but the table was created with two real columns (name and hike_safety). To fulfill your three-column specification, the function promotes only one other column, type. The next example has expanded display to list the columns vertically. Notice the ADDED status for the column that was just materialized, rather than EXISTS for the two columns you defined when creating the table:

```sql
=> \x
Expanded display is on.
=> SELECT * from materialize_flextable_columns_results where table_name = 'mountains_hybrid';

-| RECORD 1 | -------------------------------
| table_id | 45035996273766044              |
| table_schema | public                      |
| table_name | mountains_hybrid            |
| creation_time | 2013-11-30 20:09:37.765257-05 |
| key_name | type                        |
| status | ADDED                       |
| message | Added successfully           |

-| RECORD 2 | -------------------------------
| table_id | 45035996273766044              |
| table_schema | public                      |
| table_name | mountains_hybrid            |
| creation_time | 2013-11-30 20:09:37.765284-05 |
| key_name | hike_safety                  |
| status | EXISTS                       |
| message | Column of same name already exists in table definition |

-| RECORD 3 | -------------------------------
| table_id | 45035996273766044              |
| table_schema | public                      |
| table_name | mountains_hybrid            |
| creation_time | 2013-11-30 20:09:37.765296-05 |
| key_name | name                         |
| status | EXISTS                       |
| message | Column of same name already exists in table definition |
```

3. Now, display the hybrid table definition, listing the `__raw__` column and the three materialized columns. Flex table data types are derived from the associated keys tables, so you can update them as necessary. Notice that the `__raw__` column has a default NOT
You have now completed getting started with flex table basics, hybrid flex tables, and using flex functions.
Understanding Flex Tables

You can create flex tables and then manage them with their associated helper, data, and map functions. Flex tables:

- Do not require schema definitions
- Do not need column definitions
- Have full Unicode support
- Support SQL queries

You can use flex tables to promote data directly from exploration to analytic operations. Flex table features include:

- Ability to load different formats into one flex table, which lets you handle changing structure over time
- Full support of delimited and JSON data
- Extensive SQL queries and built-in analytics for the data you load
- Usability functions, which let you explore your unstructured data and then use built-in functions to materialize the data

Exploration to Promotion

After you create a flex table, you can quickly load data, including social media content in JSON, log files, delimited files, and other information. Previously, working with such data required significant schema design and preparation. Now, you can load and query flex tables in a few steps.

Creating flex tables is similar to creating other tables, except column definitions are optional. When you create flex tables, with or without column definitions, Vertica implicitly adds a real column to your table, called __raw__. This column stores loaded data. The __raw__ column is a LONG VARCHAR column with a NOT NULL constraint. It contains the documented limits for its data type (see Long Data Types in the SQL Reference Manual. The __raw__ column's default maximum width is 130,000 bytes (with an absolute maximum of 32,000,000 bytes). You can change the default width with the FlexTablesRawSize configuration parameter.
If you create a flex table without other column definitions, the table includes a second default column, __identity__, declared as an auto-incrementing IDENTITY (1,1) column. When no other columns are defined, flex tables use the __identity__ column for segmentation and sort order.

Loading data into a flex table encodes the record into a VMap type and populates the __raw__ column. The VMap is a standard dictionary type, pairing keys with string values as virtual columns.

**Flex Table Terms**

This guide uses the following terms when describing how you work with flex tables to explore and analyze flexible data:

- **VMap**: An internal map data format.
- **Virtual Columns**: Key-value pairs contained in a flex table __raw__ column.
- **Real Columns**: Fully featured columns in flex or columnar tables.
- **Promoted Columns**: Virtual columns that have been materialized to real columns.
- **Map Keys**: Map keys are the virtual column names within VMap data.

**Is There Structure in a Flex Table?**

The term *unstructured data* (sometimes called *semi-structured* or *Dark Data*) does not indicate that the data you load into flex tables is entirely without structure. However, you may not know the data's composition or the inconsistencies of its design. In some cases, the data may not be relational.

Your data may have some structure (like JSON and delimited data). Data may be semi-structured or stringently structured, but in ways that you either do not know about or do not expect. In this guide, the term *flexible data* encompasses these and other kinds of data. You can load your flexible data directly into a flex table, and query its contents with your favorite SQL SELECT or other statements.

To summarize, you can load data first, without knowing its structure, and then query its content after a few simple transformations. In some cases, you already know the data structure, such as some tweet map keys, like `user.lang`, `user.screen_name`, and `user.url`. If so, you can query these values explicitly as soon as you load the data.
Storing Flex Table Data

While you can store unstructured data in a flex table __raw__ column, that column is implemented as a real column.

Vertica compresses __raw__ column data by about one half (1/2). While this factor is less than the compression rate for real columns, the reduction is significant for large amounts (more than 1TB) of unstructured data. After compression is complete, Vertica writes the data to disk (ROS). This approach maintains K-safety in your cluster and supports standard recovery processes should node failures occur. Flex tables are included in full backups (or, if you choose, in object-level backups).

What Happens When You Create Flex Tables?

Whenever you execute a CREATE FLEX TABLE statement, Vertica creates three objects, as follows:

- The flexible table (flex_table)
- An associated keys table (flex_table_keys)
- A default view for the main table (flex_table_view)

The _keys and _view objects are dependents of the parent, flex_table. Dropping the flex table also removes its dependents, although you can drop the _keys or _view objects independently.

You can create a flex table without specifying any column definitions (such as darkdata, in the next example). When you do so, Vertica automatically creates two tables, the named flex table (such as darkdata) and its associated keys table, darkdata_keys:

```
=> CREATE flex table darkdata;
CREATE TABLE
=> \dt dark*

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_keys</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
</tbody>
</table>
(2 rows)
```
Each flex table has two default columns, __raw__ and __identity__. The __raw__ column exists in every flex table to hold the data you load. The __identity__ column is auto-incrementing. Vertica uses the __identity__ column for segmentation and sort order when no other column definitions exist. The flex keys table (darkdata_keys) has three columns, as shown:

```sql
=> SELECT * FROM darkdata;
__identity__ | __raw__
-------------
(0 rows)

=> SELECT * FROM darkdata_keys;
key_name | frequency | data_type_guess
----------|-----------|-----------------
(0 rows)
```

Creating a flex table with column definitions (such as darkdata1, in the next example) automatically generates a table with the __raw__ column. However, the table has no __identity__ column because columns are specified for segmentation and sort order. Two tables are created automatically, as shown in the following example:

```sql
=> CREATE FLEX TABLE darkdata1 (name VARCHAR);
CREATE TABLE

=> SELECT * FROM darkdata1;
__raw__ | name
------|-----
(0 rows)

=> \d darkdata1*
List of Fields by Tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata1</td>
<td><strong>raw</strong></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1</td>
<td>name</td>
<td>varchar(80)</td>
<td>80</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(2 rows)

=> \dt darkdata1*
List of tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata1</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_keys</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
</tbody>
</table>
(2 rows)
```

Creating a flex table with at least one column definition (darkdata1 in the next example) also generates a table with the __raw__ column, but not an __identity__ column. Instead, the specified columns are used for segmentation and sort order. Two tables are also created automatically, as shown in the following example:
=> CREATE FLEX TABLE darkdata1 (name VARCHAR);
CREATE TABLE
=> \d darkdata1*

<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata1</td>
<td><em>raw</em></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1</td>
<td>name</td>
<td>varchar(80)</td>
<td>80</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

=> \dt darkdata1*

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata1</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_keys</td>
<td>table</td>
<td>dbadmin</td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

For more examples, see Creating Flex Tables.

Creating Superprojections Automatically

In addition to creating two tables for each flex table, Vertica creates superprojections for both the main flex table and its associated keys table. Using the \dj command, you can display the projections created automatically for the darkdata and darkdata1 tables in this set of examples:

=> \dj darkdata*

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>List of projections</th>
<th>Owner</th>
<th>Node</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata1_b0</td>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_b1</td>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_b0</td>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_b1</td>
<td>dbadmin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>darkdata_keys_super</td>
<td>dbadmin</td>
<td>v_vmart_node0004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(10 rows)
Default Flex Table View

When you create a flex table, you also create a default view. This default view uses the table name with a _view suffix, as listed in the next example, which shows the list of views for darkdata and darkdata1. If you query the default view, you are prompted to use the COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW function. This view enables you to update the view after you load data so that it includes all keys and values.

```
=> \dv darkdata*
```

<table>
<thead>
<tr>
<th>Schema</th>
<th>View</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>darkdata_view</td>
<td>status</td>
<td>varchar(124)</td>
<td>124</td>
</tr>
<tr>
<td>public</td>
<td>darkdata1_view</td>
<td>status</td>
<td>varchar(124)</td>
<td>124</td>
</tr>
</tbody>
</table>

(2 rows)

For more information, see Updating Flex Table Views.

Flex Functions

There are three sets of functions to support flex tables and extracting data into VMaps. See the following sections for more information:

- Data (helper) functions (Flex Data Functions Reference)
- Extractor functions (Flex Extractor Functions Reference)
- Map functions (Flex Map Functions Reference)

Using Clients with Flex Tables

You can use the Vertica supported client drivers with flex tables as follows:

- To load data into a flex table, you can use the INSERT statement or COPY LOCAL with the appropriate flex table parser.

- The driver metadata APIs return only real columns. For example, using a SELECT * FROM myflex; statement, when myflex has a single materialized column (name), returns the __raw__ and name columns. However, it does not return virtual columns from within __raw__
To access virtual columns and their values, query the associated `flexttable_keys` table, just as you would in vsql.
Creating Flex Tables

You can create a flex table or an external flex table without column definitions or other parameters. You can use any CREATE TABLE statement parameters you prefer, as usual.

Unsupported CREATE FLEX TABLE Statements

These statements are not currently supported:

- `CREATE FLEX TABLE LIKE...`

Creating Basic Flex Tables

Here's how to create the table:

```
=> CREATE FLEX TABLE darkdata();
CREATE TABLE
```

Selecting from the table before loading any data into it reveals its two real columns, `__identity__` and `__raw__`:

```
=> SELECT * FROM darkdata;
__identity__  | __raw__
---------------+---------
(0 rows)
```

Below is an example of creating a flex table with a column definition:

```
=> CREATE FLEX TABLE darkdata1(name VARCHAR);
CREATE TABLE
```

When flex tables exist, you can add new columns (including those with default derived expressions), as described in Materializing Flex Tables.

Creating Temporary Flex Tables

Before you create temporary global and local flex tables, be aware of the following considerations:
GLOBAL TEMP flex tables are supported. Creating a temporary global flex table results in the flextable_keys table and the flextable_view having temporary table restrictions for their content.

LOCAL TEMP flex tables must include at least one column definition. The reason for this requirement is that local temp tables do not support automatically-incrementing data (such as the flex table default __identity__ column). Creating a temporary local flex table results in the flextable_keys table and the flextable_view existing in the local temporary object scope.

LOCAL TEMP views are supported for flex and columnar temporary tables.

For global or local temp flex tables to function correctly, you must also specify the ON COMMIT PRESERVE ROWS clause. You must use the ON COMMIT clause for the flex table helper functions, which rely on commits. Create a local temp table as follows:

```sql
=> CREATE FLEX LOCAL TEMP TABLE good(x int) ON COMMIT PRESERVE ROWS;
CREATE TABLE
```

After creating a local temporary flex table using this approach, you can then load data into the table, create table keys, and a flex table view:

```sql
=> COPY good FROM '/home/release/KData/bake.json' PARSER fjsonparser();
Rows Loaded
----------
 1
(1 row)
=> select compute_flextable_keys_and_build_view('good');
  compute_flextable_keys_and_build_view
----------
Please see v_temp_schema.good_keys for updated keys
The view good_view is ready for querying
(1 row)
```

Similarly, you can create global temp tables as follows:

```sql
=> CREATE FLEX GLOBAL TEMP TABLE good_global(x int) ON COMMIT PRESERVE ROWS;
```

After creating a global temporary flex table using this approach, you can then load data into the table, create table keys, and a flex table view:

```sql
=> COPY good_global FROM '/home/dbadmin/data/flex/bake_single.json' PARSER fjsonparser();
Rows Loaded
---------
  5
(1 row)
=> SELECT compute_flextable_keys_and_build_view('good_global');
```
Materializing Flex Table Virtual Columns

After you create your flex table and load data, you compute keys from virtual columns. After completing those tasks, you can materialize some keys by promoting virtual columns to real table columns. By promoting virtual columns, you query real columns rather than the raw data.

You can promote one or more virtual columns — materializing those keys from within the ___raw___ data to real columns. Vertica recommends this approach to get the best query performance for all important keys. You don't need to create new columnar tables from your flex table.

Materializing flex table columns results in a hybrid table. Hybrid tables:

- Maintain the convenience of a flex table for loading unstructured data
- Improve query performance for any real columns

If you have only a few columns to materialize, try altering your flex table progressively, adding columns whenever necessary. You can use the ALTER TABLE . . . ADD COLUMN statement to do so, just as you would with a columnar table. See Materializing Flex Tables for ideas about adding columns.

If you want to materialize columns automatically, use the helper function MATERIALIZE_FLEXTABLE_COLUMNS

Creating Columnar Tables from Flex Tables

You can create a regular Vertica table from a flex table, but you cannot use one flex table to create another.

Typically, you create a columnar table from a flex table after loading data. Then, you specify the virtual column data you want in a regular table, casting virtual columns to regular data types.
To create a columnar table from a flex table, `darkdata`, select two virtual columns, `user.lang` and `user.name`, for the new table. Use a command such as the following, which casts both columns to `varchars` for the new table:

```
=> CREATE TABLE darkdata_full AS SELECT "user.lang"::VARCHAR, "user.name"::VARCHAR FROM darkdata;
```

```
CREATE TABLE
=> SELECT * FROM darkdata_full;
```

<table>
<thead>
<tr>
<th>user.lang</th>
<th>user.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>Frederick Danjou</td>
</tr>
<tr>
<td>en</td>
<td>The End</td>
</tr>
<tr>
<td>en</td>
<td>Uptown gentleman.</td>
</tr>
<tr>
<td>en</td>
<td>~G A B R I E L A â¿</td>
</tr>
<tr>
<td>es</td>
<td>Flu Beach</td>
</tr>
<tr>
<td>es</td>
<td>I'm Toasterâ¥</td>
</tr>
<tr>
<td>it</td>
<td>laughing at clouds.</td>
</tr>
<tr>
<td>tr</td>
<td>seydo shi</td>
</tr>
</tbody>
</table>

(12 rows)

### Creating External Flex Tables

To create an external flex table:

```
=> CREATE flex external table mountains() AS COPY FROM 'home/release/KData/kmm_ountains.json' PARSER fjsonparser();
```

As with other flex tables, creating an external flex table produces two regular tables: the named table and its associated `_keys` table. The keys table is not an external table:

```
=> \dt mountains
```

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>mountains</td>
<td>table</td>
<td>release</td>
<td>(1 row)</td>
</tr>
</tbody>
</table>

You can use the helper function, `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`, to compute keys and create a view for the external table:

```
=> SELECT compute_flextable_keys_and_build_view('appLog');
```

```
```

Please see `public.appLog_keys` for updated keys

The view `public.appLog_view` is ready for querying
1. Check the keys from the _keys table for the results of running the helper application:

```sql
=> SELECT * FROM appLog_keys;

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>coordinates</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>created_at</td>
<td>8</td>
<td>varchar(60)</td>
</tr>
<tr>
<td>entities.hashtags</td>
<td>8</td>
<td>long varbinary(186)</td>
</tr>
<tr>
<td>retweeted_status.user.time_zone</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>retweeted_status.user.url</td>
<td>1</td>
<td>varchar(68)</td>
</tr>
<tr>
<td>retweeted_status.user.utc_offset</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>retweeted_status.user.verified</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
</tbody>
</table>
```

(125 rows)

2. Query from the external flex table view:

```sql
=> SELECT "user.lang" FROM appLog_view;

<table>
<thead>
<tr>
<th>user.lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>it</td>
</tr>
<tr>
<td>en</td>
</tr>
<tr>
<td>es</td>
</tr>
<tr>
<td>en</td>
</tr>
<tr>
<td>es</td>
</tr>
<tr>
<td>tr</td>
</tr>
<tr>
<td>en</td>
</tr>
</tbody>
</table>
```

(12 rows)

Note: External tables are fully supported for both flex and columnar tables. However, using external flex (or columnar) tables is less efficient than using flex tables whose data is stored in the Vertica database. Data that is maintained externally requires reloading each time you query.

### Creating a Flex Table from Query Results

You can use the CREATE FLEX TABLE AS statement to create a flex table from the results of a query.

You can use this statement to create three types of flex tables:
- Flex table with no materialized columns
- Flex table with some materialized columns
- Flex table with all materialized columns

When a flex __raw__ column is present in the CTAS query, the entire source VMap is carried to the flex table. If the query has matching column names, the key values are overridden.

Note: ORDER BY and segmentation clauses are only passed to the new flex table if the relevant columns are materialized.

Examples

Creating a flex table with no materialized columns from a regular table causes the results of the query to be stored in the __raw__ column as a VMap.

1. Create a regular table named pets with two columns:

   ```sql
   => CREATE TABLE pets(age INT, name VARCHAR);
   CREATE TABLE
   ```

2. Create a flex table named family_pets by using the CTAS statement to copy the columns age and name from the pets:

   ```sql
   => CREATE FLEX TABLE family_pets() AS SELECT age, name FROM pets;
   CREATE TABLE
   ```

3. View the new flex table to confirm the operation has been successful and that the columns age and name have not been materialized.

   ```sql
   => \d family_pets;
   List of Fields by Tables
   Schema | Table | Column | Type | Size | Default | Not Null |
   Primary Key | Foreign Key
   ------+--------+---------+-------+-----+--------+---------|
   public | family_pets | __identity__ | int | 8 | t | f
   | | | | | | |
   public | family_pets | __raw__ | long varbinary(130000) | 130000 | t | f
   | | | | | | |
   (2 rows)
   ```
You can create a flex table with no materialized columns from the results of a query of another flex table. This inserts all the VMaps from the source flex table into the target. This creates a flex table segmented and ordered by the __identity__ column.

4. Create a flex table named city_pets by using the CTAS statement to copy the age and __raw__ columns from family_pets:

```sql
=> CREATE FLEX TABLE city_pets() AS SELECT age, __raw__ FROM family_pets;
```

5. View the new flex table to confirm that the operation has been successful and the columns age and __raw__ have not been materialized.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>city_pets</td>
<td><strong>identity</strong></td>
<td>int</td>
<td>8</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>city_pets</td>
<td><strong>raw</strong></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td>f</td>
</tr>
</tbody>
</table>

You can create a flex table with some materialized columns. This uses a syntax similar to the syntax for creating columnar tables with some materialized columns. Unlike columnar tables, however, you need to match the number of columns with the columns that are returned by the query. In the following example, our query returns three columns (amount, type, and available), but Vertica only materializes the first two.

6. Create a table named animals with three columns, amount, type, and available:

```sql
=> CREATE TABLE animals(amount INT, type VARCHAR, available DATE);
```

7. Create a flex table named inventory with columns animal_amount and animal_type using the CTAS statement to copy columns amount, type, and available from animals.

```sql
=> CREATE FLEX TABLE inventory(animal_amount, animal_type) AS SELECT amount, type, available FROM animals;
```

8. View the table data to confirm that columns amount and type have been materialized under the column names animal_amount and animal_type. Column available from animals has also been copied over but was not materialized:
=> \d inventory
List of Fields by Tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>flex3</td>
<td><strong>raw</strong></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>flex3</td>
<td>animal_amount</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>flex3</td>
<td>animal_type</td>
<td>varchar(80)</td>
<td>80</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>
(3 rows)

Notice that including empty parentheses in the statement results in a flex table with no materialized columns:

9. Create a flex table named animals_for_sale using the CTAS statement with empty parentheses to copy columns amount, type, and available from animals into a pure flex table:

=> CREATE FLEX TABLE animals__for_sale() AS SELECT amount, type, available FROM animals;
CREATE TABLE

10. View the table data to confirm that no columns were materialized:

=> \d animals_for_sale;
List of Fields by Tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>animals_for_sale</td>
<td><strong>identity</strong></td>
<td>int</td>
<td>8</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>raw</strong></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td></td>
</tr>
</tbody>
</table>
(2 rows)

Omitting any parentheses in the statement causes all columns to be materialized:

11. Create a flex table named animals_sold using the CTAS statement without parentheses. This copies columns amount, type, and available from animals and materialize all columns:

=> CREATE FLEX TABLE animals_sold AS SELECT amount, type, available FROM animals;
CREATE TABLE

12. View the table data to confirm that all columns were materialized:

=> \d animals_sold;
List of Fields by Tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>animals_sold</td>
<td><strong>raw</strong></td>
<td>long varbinary(130000)</td>
<td>130000</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>animals_sold</td>
<td>amount</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>animals_sold</td>
<td>type</td>
<td>varchar(80)</td>
<td>80</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>public</td>
<td>animals_sold</td>
<td>available</td>
<td>date</td>
<td>8</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

(4 rows)
Bulk Loading Data into Flex Tables

You bulk load data into a flex table with a COPY statement, specifying one of the flex parsers:

- FAVROPARSER
- FCEFPARSER
- FCSVPARSER
- FDELIMITEDPAIRPARSER
- FDELIMITEDPARSER
- FJSONPARSER
- FREGEXPARSER

All flex parsers store the data as a single-value VMap. They reside in the VARBINARY __raw__ column, which is a real column with a NOT NULL constraint. The VMap is encoded into a single binary value for storage in the __raw__ column. The encoding places the value strings in a contiguous block, followed by the key strings. Vertica supports null values within the VMap for keys with NULL-specified columns. The key and value strings represent the virtual columns and their values in your flex table.

If a flex table data row is too large to fit in the VARBINARY __raw__ column, it is rejected. By default, the rejected data and exceptions files are stored in the standard CopyErrorLogs location, a subdirectory of the catalog directory:

```
v_mart_node003_catalog/CopyErrorLogs/trans-STDIN-copy-from-exceptions.1
v_mart_node003_catalog/CopyErrorLogs/trans-STDIN-copy-rejections.1
```

Flex tables do not copy any rejected data, due to disk space considerations. The rejected data file exists, but it contains only a new line character for every rejected record. The corresponding exceptions file lists the reason why each record was rejected.

You can specify a different path and file for the rejected data and exceptions files. To do so, use the COPY parameters REJECTED DATA and EXCEPTIONS, respectively. You can also save load rejections and exceptions in a table. For more information, see Bulk-Loading Data.
Basic Flex Table Load and Query

Loading data into your flex table is similar to loading data into a regular columnar table. The difference is that you must use the `parser` argument with one of the flex parsers:

```
=> COPY darkdata FROM '/home/dbadmin/data/tweets_12.json' PARSER fjsonparser;
Rows Loaded
---------
  12
(1 row)
```

Note: You can use many additional COPY parameters as required but not all are supported.

Loading Data into Flex Table Real Columns

If you create a hybrid flex table with one or more real column definitions, COPY evaluates each virtual column key name during data load. For each real column with a name that is identical to a virtual column key name, COPY does the following:

- Loads the keys and values as part of the VMap data in the `__raw__` column
- Automatically populates real columns with the values from their virtual column counterparts

Subsequent data loads continue loading same-name key-value pairs into both the `__raw__` column and the real column.

Note: Over time, storing values in both column types can impact your licensed data limits. For more information about Vertica licenses, see Managing Licenses in the Administrator's Guide.

For example, continuing with the JSON data:

1. Create a flex table, `darkdata1`, with a column definition of one of the keys in the data you will load:

```
=> CREATE FLEX TABLE darkdata1 ("user.lang" VARCHAR);
CREATE TABLE
```
2. **Load data into darkdata1:**

```sql
=> COPY darkdata1 FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-----------
 12
(1 row)
```

3. **Query the user.lang column of darkdata1. Loading the JSON data file populated the column you defined:**

```sql
=> SELECT "user.lang" FROM darkdata1;
    user.lang
----------
   es
   es
   tr
   it
   en
   en
   en
   (12 rows)
```

Empty column rows indicate NULL values. For more information about how NULLs are handled in flex tables, see [NULL Value](#).

4. **You can query for other virtual columns (such as "user.name" in darkdata1), with similar results as for "user.lang":**

```sql
=> SELECT "user.name" FROM darkdata1;
     user.name
-------------------
  I'm Toasterâ¥ Flu Beach seydo shi The End Uptown gentleman. ~G A B R I E L A â¿ Frederick Danjou laughing at clouds. (12 rows)
```

**Note:** While the results for these two queries are similar, the difference in accessing the keys and their values is significant. Data for "user.lang" has been materialized into a real table column, while "user.name" remains a virtual column. For production-level data usage (rather than test data sets), materializing flex table data improves query performance significantly.
Handling Default Values During Loading

You can create your flex table with a real column, named for a virtual column that exists in your incoming data. For example, if the data you load has a `user.lang` virtual column, define the flex table with that column. You can also specify a default column value when creating the flex table with real columns. The next example shows how to define a real column (`user.lang`), which has a default value from a virtual column (`user.name`):

```sql
CREATE FLEX TABLE darkdata1 ("user.lang" LONG VARCHAR default "user.name");
```

When you load data into your flex table, COPY uses values from the flex table data, ignoring the default column definition. Loading data into a flex table requires MAPLOOKUP to find keys that match any real column names. A match exists when the incoming data has a virtual column with the same name as a real column. When COPY detects a match, it populates the column with values. COPY returns either a value or NULL for each row, so real columns always have values.

For example, after creating the `darkdata1` flex table, described in the previous example, load data with COPY:

```sql
COPY darkdata1 FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
----------
  12
(1 row)
```

If you query the `darkdata1` table after loading, the data shows that the values for the `user.lang` column were extracted:

- From the data being loaded — values for the `user.lang` virtual column
- With NULL — rows without values

In this case, the table column default value for `user.lang` was ignored:

```sql
SELECT "user.lang" FROM darkdata1;
user.lang
----------
it
en
es
en
es
tr
```
Using COPY to Specify Default Column Values

You can add an expression to a COPY statement to specify default column values when loading data. For flex tables, specifying any column information requires that you list the _raw column explicitly. The following example shows how to use an expression for the default column value. In this case, loading populates the defined user.lang column with data from the input data user.name values:

```sql
=> COPY darkdata1(_raw_, "user.lang" as "user.name"::VARCHAR)  
   FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
Rows Loaded
-------------
  12
(1 row)
=> SELECT "user.lang" FROM darkdata1;
  user.lang
------------------------
laughing at clouds.
Avita Desai
I'm Toasterâ€’
Uptown gentleman.
~G A B R I E L A â¿
Flu Beach
seydo shi
The End
(12 rows)
```

You can specify default values when adding columns, as described in Altering Flex Tables. When you do so, a different behavior results. For more information about using COPY, its expressions and parameters, see Bulk-Loading Data in the Administrator's Guide and COPY in the SQL Reference Manual.
Inserting Data into Flex Tables

You can load data into a Vertica flex table using a standard INSERT statement, specifying data for one or more columns. When you use INSERT, Vertica populates any materialized columns and stores the VMap data in the __raw__ column.

Vertica provides two ways to use INSERT with flex tables:

- INSERT ... VALUES
- INSERT ... SELECT

Inserting Values into Flex Tables

To insert data values into a flex table, use an INSERT ... VALUES statement. If you do not specify any columns in your INSERT ... VALUES statement, Vertica positionally assigns values to the real columns of the flex table.

This example shows two ways to insert values into a simple flex table. For both statements, Vertica assigns the values 1 and 'x' to columns a and b, respectively. This example inserts values into the two real columns defined in the flex table:

```
=> CREATE FLEX TABLE flex0 (a INT, b VARCHAR);
CREATE TABLE
=> INSERT INTO flex0 VALUES (1, 'x');
OUTPUT
-------
  1
(1 row)
```

This example inserts values into a flex table without any real columns:

```
=> CREATE FLEX TABLE flex1();
CREATE TABLE
=> INSERT INTO flex1(a,b) VALUES (1, 'x');
OUTPUT
-------
  1
(1 row)
```

For the preceding example, the __raw__ column contains the inserted data:

```
=> SELECT MapToString(__raw__) FROM flex1;
MapToString
-----------------
{ }
```
Using INSERT ... SELECT with Flex Tables

Using an INSERT ... SELECT statement with a flex table is similar to using INSERT ... SELECT with a regular table. The SELECT statement returns the data to insert into the target table.

However, Vertica does not require that you balance the number of columns and values. If you do not specify a value for a column, Vertica inserts NULL.

In the next example, Vertica copies the a and b values from the flex1 table, and creates columns c, d, e, and f. Because the statement does not specify a value for f, Vertica assigns it a NULL.

```sql
=> CREATE FLEX TABLE flex2();
CREATE TABLE
=> INSERT INTO flex2(a,b) SELECT a,b, '2016-08-10 11:10' c, 'Hello' d, 3.1415 e, f from flex1;
OUTPUT
--------
1
(1 row)
=> SELECT MapToString(__raw__) FROM flex2;
MapToString

-------------------------

{
  "a" : "1",
  "b" : "x",
  "c" : "2016-08-10 11:10",
  "d" : "Hello",
  "e" : "3.1415",
  "f" : null
}
(1 row)
```

Inserting __raw__ Columns into a Flex Table

Inserting a __raw__ column into a flex table inserts the entire source VMap into the target table. Vertica does not assign the __raw__ column to any target column. Its position in the SELECT statement does not matter.

The following two INSERT statements are equivalent.
Error Handling

Type coercion errors occur only with real columns. The insert operation fails as follows:

```
=> INSERT INTO flex4(a,b) SELECT a, __raw__, b FROM flex3;
=> INSERT INTO flex4(a,b) SELECT a, b, __raw__ FROM flex3;
```

If you try to insert values into the __raw__ column, the insert fails as follows:

```
=> CREATE FLEX TABLE my_table(a INT, b VARCHAR);
CREATE TABLE
=> INSERT INTO my_table(a, b) VALUES ('xyz', '5');
ERROR: Invalid input syntax for integer: "xyz"
```

See Also

- INSERT
- Bulk Loading Data into Flex Tables
- Data Type Coercion
Using Flex Tables for IDOL Data

You can create flex tables to use with the IDOL Connector Framework Server (CFS) and an ODBC client. The CFS VerticalIndexer module uses the connector to retrieve data. CFS then indexes the data into your Vertica database.

CFS supports many connectors for interfacing to different unstructured file types stored in repositories. Examples of repositories include Microsoft Exchange (email), file systems (including Word documents, images, and videos), Microsoft SharePoint, and Twitter (containing Tweets).

Connectors retrieve and aggregate data from repositories. CFS indexes the data, sending it to IDOL, IDOL OnDemand, or Vertica. The following figure illustrates a basic setup with a repository and a connector.

After you configure CFS and connect it to your Vertica database, the connector monitors the repository for changes and deletions to loaded documents, and for new files not previously added to the server. CFS then updates its server destinations automatically.

To achieve the best query results with ongoing CFS updates and deletes, Vertica recommends using live aggregate projections and top-K projections. For more information about how these projections work, and for examples of using them, see Working with Projections in the Administrator's Guide.

ODBC Connection String for CFS

There are several steps to setting up the CFS VerticalIndexer to load IDOL metadata into your database.
One of the first steps is to add information to the CFS configuration file. To do so, add an Indexing section to the configuration file that specifies the ODBC ConnectionString details.

Successfully loading data requires a valid database user with write permissions to the destination table. Two ODBC connection parameters (UID and PWD) specify the Vertica user and password. The following example shows a sample CFS Indexing section. The section includes a ConnectionString with the basic parameters, including a sample user (UID=fjones) and password (PWD=fjones_password):

```plaintext
[Indexing]
IndexerSections=vertica
IndexTimeInterval=30
[vertica]
IndexerType = Library
ConnectionString=Driver=Vertica;Server=123.456.478.900;Database=myDB;UID=fjones;PWD=fjones_password
TableName = marcomm.myFlexTable
LibraryDirectory = ./shared_library_indexers
LibraryName = verticaIndexer
```

For more information about ODBC connection parameters, see ODBC Configuration Parameters.

**CFS COPY LOCAL Statement**

CFS first indexes and processes metadata from a document repository to add to your database. Then, CFS uses the Indexing information you added to the configuration file to create an ODBC connection. After establishing a connection, CFS generates a standard COPY LOCAL statement, specifying the fjsonparser. CFS loads data directly into your pre-existing flex table with a statement such as the following:

```plaintext
=> COPY myFlexTable FROM LOCAL path_to_compressed_temporary_json_file PARSER fjsonparser();

=> SELECT * FROM myavro;
  __identity__ | __raw__
  ------------------------
  (0 rows)
```

When your IDOL metadata appears in a flex table, you can optionally add new table columns, or materialize other data, as described in Altering Flex Tables.
Using Flex Table Parsers

You can load flex tables with one of several parsers. You can load data using the options that the flex parsers support:

- Loading Avro Data
- Loading Common Event Format (CEF) Data
- Loading CSV Data
- Loading Delimited Data
- Loading JSON Data
- Loading Matches from Regular Expressions

Using Flex Parsers for Columnar Tables

You can use any of the flex parsers to load data into columnar tables. Using the flex table parsers to load columnar tables gives you the capability to mix data loads in one table. For example, you can load JSON data in one session and delimited data in another.

Note: For Avro data, you can load only data into a columnar table, not the schema. For flex tables, Avro schema information is required to be embedded in the data.

The following basic examples illustrate how you can use flex parsers with columnar tables.

1. Create a columnar table, `super`, with two columns, age and name:

   ```sql
   => CREATE TABLE super(age INT, name VARCHAR);
   CREATE TABLE
   ```

2. Enter JSON values from STDIN, using the `fjsonparser()`:

   ```sql
   => COPY super FROM stdin PARSER fjsonparser();
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
   >> {"age": 5, "name": "Tim"}
   >> {"age": 3}
   >> {"name": "Fred"}
   >> {"name": "Bob", "age": 10}
   ```
3. Query the table to see the values you entered:

```sql
SELECT * FROM super;
```

```
+----+-----+
| age| name |
|----+-----|
| 10 | Bob  |
|  5 | Tim  |
|  3 |     |
(4 rows)
```

4. Enter some delimited values from STDIN, using the `fdelimitedparser()`:

```sql
COPY super FROM stdin PARSER fdelimitedparser();
```

```
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> name | age
>> Tim |50
>>  |30
>> Fred |
>> Bob |100
>> 
```

5. Query the flex table. Both JSON and delimited data are saved in the same columnar table, `super`.

```sql
SELECT * FROM super;
```

```
+----+-----+
| age| name |
|----+-----|
| 50 | Tim  |
| 30 |     |
|  5 | Tim  |
| 100| Bob  |
|    | Fred |
|    | Bob  |
|    | Fred |
(8 rows)
```

Use the `reject_on_materialized_type_error` parameter to avoid loading data with type mismatch. If `reject_on_materialized_type_error` is set to `false`, the flex parser will accept the data with type mismatch. Consider the following example:

Assume that the CSV file to be loaded has the following sample contents:

```sh
$ cat json.dat
{"created_by":"system","site_source":"flipkart_india_kol","updated_by":"system1","invoice_id":"INVDPKOL100","vendor_id":"VEN15731","total_quantity":12,"created_at":"2012-01-09 23:15:52.0"}
```
1. Create a columnar table.

```sql
=> CREATE TABLE hdfs_test (  
site_source VARCHAR(200),
total_quantity int,
vendor_id varchar(200),
invoice_id varchar(200),
updated_by varchar(200),
created_by varchar(200),
created_at timestamp  
);
```

2. Load JSON data.

```sql
=> COPY hdfs_test FROM '/home/dbadmin/json.dat' PARSER fjsonparser() ABORT ON ERROR;  
Rows Loaded  
--------------  
2  
(1 row)
```

3. View the contents.

```sql
=> SELECT * FROM hdfs_test;  
site_source | total_quantity | vendor_id | invoice_id | updated_by | created_by | created_at  
-------------- |-------------- |---------- |---------- |---------- |---------- |----------  
flipkart_india_kol | 12 | VEN15731 | INVDPKOL100 | system1 | system | 2012-01-09 23:15:52  
flipkart_india_kol | 14 | VEN15732 | INVDPKOL101 | system2 | system | 2012-01-09 23:15:52  
(2 rows)
```

4. If `reject_on_materialized_type_error` parameter is set to true, you will receive errors when loading the sample JSON data.

```sql
=> COPY hdfs_test FROM '/home/dbadmin/data/flex/json.dat' PARSER fjsonparser(reject_on_materialized_type_error=true) ABORT ON ERROR;  
ERROR 2035:  COPY: Input record 2 has been rejected (Rejected by user-defined parser)
```

### Loading Avro Data

You can load Avro data files into flex tables and columnar tables using the parser, `favroparser`. Before loading, verify that Avro files are encoded in the Avro binary serialization encoding format, described in the [Apache Avro standard](https://avro.apache.org). The parser also supports Snappy compression. You cannot load Avro data directly from STDIN.
Note: The parser favroparser does not support Avro files with separate schema files. The Avro file must have its related schema in the file you are loading.

You can use the following data types and optional parameters for favroparser.

The favroparser supports two data types:

- **Primitive Data Types for favroparser**
- **Complex Data Types for favroparser**

### Rejecting Data on Materialized Column Type Errors

The favroparser has a Boolean parameter, `reject_on_materialized_type_error`. If you set this parameter to true, Vertica rejects rows when the input data presents both of the following conditions:

- Includes keys matching an existing materialized column
- Has a value that cannot be coerced into the materialized column's data type

Suppose the flex table has a materialized column, Temperature, declared as a FLOAT. If you try to load a row with a Temperature key that has a VARCHAR value, favroparser rejects the data row.

### See Also

- [Manually Copying Data From Kafka](#)

### Primitive Data Types for favroparser

The favroparser supports the following primitive data types:

<table>
<thead>
<tr>
<th>AVRO Data Type</th>
<th>Vertica Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL Value</td>
<td>No value</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean Data Type</td>
<td>A binary value</td>
</tr>
<tr>
<td>int</td>
<td>INTEGER</td>
<td>32-bit</td>
</tr>
<tr>
<td>Type</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>long</td>
<td>INTEGER</td>
<td>64-bit signed integer</td>
</tr>
<tr>
<td>float</td>
<td>DOUBLE PRECISION (FLOAT)</td>
<td>Single precision (32-bit) IEEE 754 floating-point number</td>
</tr>
<tr>
<td>double</td>
<td>DOUBLE PRECISION (FLOAT)</td>
<td>Double precision (64-bit) IEEE 754 floating-point number</td>
</tr>
<tr>
<td>bytes</td>
<td>BYTES</td>
<td>Sequence of 8-bit unsigned bytes</td>
</tr>
<tr>
<td>string</td>
<td>VARCHAR</td>
<td>Unicode character sequence</td>
</tr>
</tbody>
</table>

**Note:** Vertica does not have an explicit 4-byte (32-bit integer) or smaller types. Instead, Vertica encoding and compression automatically eliminate the storage overhead of values that require less than 64 bits.

Vertica copies each primitive type into the `raw` column of the flex table. In this copy operation, the name of the primitive type becomes a virtual column key with its corresponding value as the value of the virtual column.

If the flex table has materialized columns, `favroparser` loads the primitive data type into the corresponding Vertica type for the column. If the parsing is successful, Vertica copies the data into the materialized column; otherwise, it rejects the row.
Complex Data Types for favroparser

You specify the data type of a record in the Avro file using the `type` parameter for favroparser. The favroparser supports these complex data types:

- Records
- Enums
- Arrays
- Maps
- Unions
- Fixed

This section describes attributes associated with the complex data types.

Records

Records have the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A JSON string for the name of the record</td>
</tr>
</tbody>
</table>
| fields    | A JSON array used to list fields. Each field is a JSON object:  
  - name: A JSON string for the name of the field  
  - type: A JSON object used to define a schema or a JSON string used for naming a record definition |

The name of each field is used as a virtual column name. If `flatten_records = true` and several nesting levels are present, Vertica concatenates the record names to create the `key_name`, as follows:
Vertica creates virtual columns for the records as follows:

<table>
<thead>
<tr>
<th>Names</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>VerticaUser</td>
</tr>
<tr>
<td>Address</td>
<td>VerticaUser Address</td>
</tr>
</tbody>
</table>

**Enums**

Enums (enumerated values) use the type name `enum` and support the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A JSON string for the name of the enum</td>
</tr>
<tr>
<td>symbols</td>
<td>A JSON array used to list symbols as JSON strings. All symbols in an enum must be unique and duplicates are prohibited</td>
</tr>
</tbody>
</table>

**Example:**

```json
{
  "type": "enum",
  "name": "suit",
  "symbols": ["SPADES", "HEARTS", "DIAMONDS", "CLUBS"]
}
```
Consider the preceding Avro schema with a record that contains a field with the value HEARTS. In this case, the key value pair copied into the _raw__ column has suit as the key and HEARTS as the value.

Arrays

Arrays use the type name array and support one attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>The schema of the array's items</td>
</tr>
</tbody>
</table>

For example, declare an array of strings:

```
{"type": "array", "items": "string"}
```

Similar to the capabilities for Records, you can nest and flatten Arrays using flatten_arrays=true:

```
{
    "_name_" : "Order",       <-- artificial _name__ key for record
    "customer_id" : "111222",
    "order_details" : {       <-- array of records
        "0" : {             <-- array index 0
            "_name_" : "OrderDetail",
            "product_detail" : {
                "_name_" : "Product",
                "price" : "46.21",
                "product_category" : {       <-- array of strings
                    "0" : "electronics",
                    "1" : "printers",
                    "2" : "computers"
                },
                "product_name" : "hp printer X11ew",
                "product_status" : "ONLY_FEW_LEFT"
            }
        },
        "order_id" : "2389646",
        "total" : "132.43"
    }
}
```

Here is the result of flattening the array:

```
{
    "0.order_details._name_" : "OrderDetail",
    "0.order_details.product_detail.0.product_category" : "electronics",
    "0.order_details.product_detail.1.product_category" : "printers",
    "0.order_details.product_detail.2.product_category" : "computers",
    "0.order_details.product_detail._name_" : "Product",
    "0.order_details.product_detail.price" : "46.21",
    "0.order_details.product_detail.product_name" : "hp printer X11ew",
    ...
Maps

Maps use the type name map and support one attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>The schema of the map's items</td>
</tr>
</tbody>
</table>

The *favroparser* treats map keys as strings. For example, you can declare the map type as a long as follows:

```json
{"type": "map", "values": "long"}
```

Similar to Records types, Maps can also be nested and flattened using `flatten_maps=true`.

The *favroparser* inserts key-value pairs from the Avro map as key-value pairs in the `__raw__` column. For an Avro record that has KeyX with value 10, and KeyY with value 20, *favroparser* loads the key-value pairs as virtual columns KeyX and KeyY, with values 10 and 20, respectively.

Unions

Vertica uses JSON arrays to represent Avro Unions. Consider this example:

```json
{"name":"TransactionID","type:["string","null"]}
```

The field TransactionID can be a string or null.

Fixed

Fixed *(fixed)* Avro types support two attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A string for the name of this</td>
</tr>
</tbody>
</table>
For example, you can declare a 16-byte quantity:

```json
{"type": "fixed", "size": 16, "name": "md5"}
```

With the preceding declaration is the Avro file schema, consider a record that contains a field with the following byte values for the key `md5`:

```plaintext
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
```

The `favroparser` loads the key value pair as an md5 key with the preceding byte values.

### Loading Common Event Format (CEF) Data

Use the flex parser `fcefparser` to load OpenText ArcSight or other Common Event Format (CEF) log file data into columnar and flexible tables. For more information, see the ArcSight Common Event Format (CEF) Guide.

When you use the parser to load arbitrary CEF-format files, it interprets key names in the data as virtual columns in your flex table. After loading, you can query your CEF data directly, regardless of which set of keys exist in each row. You can also use the associated flex table data and map functions to manage CEF data access.

### Create a Flex Table and Load CEF Data

This section uses a sample set of CEF data. All IP addresses have been purposely changed to be inaccurate, and Return characters added for illustration.

To use this sample data, copy the following text and remove all Return characters. Save the file as `CEF_sample.cef`, which is the name used throughout these examples.
1. Create a flex table logs:

```sql
> CREATE FLEX TABLE logs();
CREATE TABLE
```

2. Load the sample CEF file, using the flex parser fcefparser:

```sql
> COPY logs FROM '/home/dbadmin/data/CEF_sample.cef' PARSER fcefparser();
Rows Loaded
------------------
1
(1 row)
```

3. Use the maptostring() function to see the contents of the logs flex table:

```sql
> SELECT maptostring(__raw__) FROM logs;
maptostring

--------------------------------------------------------------------------------------
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;_cefver&quot; : &quot;0.1&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;agentzoneuri&quot; : &quot;/All Zones/ArcSight System/Private Address Space Zones/RFC1918: 888.99.0.0-888.200.255.255&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;agt&quot; : &quot;888.99.100.1&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;ahost&quot; : &quot;SKEELES10&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;aid&quot; : &quot;3DxX1G0UBABCAAA0cXXAZIwa==&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;art&quot; : &quot;1396328241038&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;at&quot; : &quot;testalertng&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;atz&quot; : &quot;Australia/Sydney&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;av&quot; : &quot;6.0.3.6664.0&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;c6a4&quot; : &quot;fe80::0:0:0:495d:cc3:db1a:de71&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;c6a4label&quot; : &quot;Agent IPv6 Address&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;cat&quot; : &quot;/Agent/Started&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;catdt&quot; : &quot;Security Mangement&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;categorybehavior&quot; : &quot;/Execute/Start&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;categorydevicegroup&quot; : &quot;/Application&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;categoryobject&quot; : &quot;/Host/Application/Service&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;categoryoutcome&quot; : &quot;/Success&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;categorysignificance&quot; : &quot;/Normal&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;cs2&quot; : &lt;Resource ID=&quot;3DxX1G0UBABCAAA0cXXAZIwa==&quot;/&gt;&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;cs2label&quot; : &quot;Configuration Resource&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;deviceproduct&quot; : &quot;ArcSight&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;deviceseverity&quot; : &quot;Warning&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;devicevendor&quot; : &quot;ArcSight&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;deviceversion&quot; : &quot;6.0.3.6664.0&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;devicezoneuri&quot; : &quot;/All Zones/ArcSight System/Private Address Space Zones/RFC1918: 888.99.0.0-888.200.255.255&quot;,</td>
<td></td>
</tr>
<tr>
<td>&quot;dtz&quot; : &quot;Australia/Sydney&quot;,</td>
<td></td>
</tr>
</tbody>
</table>
"dvc" : "888.99.100.1",
"dvchost" : "SKEELES10",
"eventid" : "1",
"filetype" : "Agent",
"mrt" : "1396328238973",
"name" : "Agent [test] type [testalertng] started",
"rt" : "1396328238937",
"severity" : "Low",
"signatureid" : "agent:030",
"version" : "0"
}
(1 row)

Create a Columnar Table and Load CEF Data

This example lets you compare the flex table for CEF data with a columnar table. You do so by creating a new table and load the same CEF_sample.cef file used in the preceding flex table example.

1. Create a columnar table, col_logs, defining the prefix names that are hard coded in fcefparser:

```sql
=> CREATE TABLE col_logs(version INT,
   devicevendor VARCHAR,
   deviceproduct VARCHAR,
   deviceversion VARCHAR,
   signatureid VARCHAR,
   name VARCHAR,
   severity VARCHAR);
CREATE TABLE
```

2. Load the sample file into col_logs, as you did for the flex table:

```sql
=> COPY col_logs FROM '/home/dbadmin/data/CEF_sample.cef' PARSER fcefparser();
Rows Loaded
----------
  1
(1 row)
```

3. Query the table. You can find the identical information in the flex table output.

```sql
=> \x
Expanded display is on.
VMart=> SELECT * FROM col_logs;
- [ RECORD 1 ]---------------------------------------------
  version | 0
  devicevendor | ArcSight
```
Compute Keys and Build a Flex Table View

In this example, you use a flex helper function to compute keys and build a view for the logs flex table.

1. Use the `compute_flextable_keys_and_build_view` function to compute keys and populate a view generated from the logs flex table:

```sql
=> SELECT compute_flextable_keys_and_build_view('logs');
```

```
Please see public.logs_keys for updated keys
The view public.logs_view is ready for querying
(1 row)
```

2. Query the logs_keys table to see what the function computed from the sample CEF data:

```sql
=> SELECT * FROM logs_keys;
```

```
key_name    | frequency | data_type_guess
-----------|-----------|------------------
c6a4        | 1         | varchar(60)      
c6a4label   | 1         | varchar(36)      
categoryobject | 1   | varchar(50)      
categoryoutcome | 1  | varchar(20)      
categoriesignificance | 1 | varchar(20)     
cs2         | 1         | varchar(84)      
cs2label    | 1         | varchar(44)      
deviceproduct | 1   | varchar(20)      
deviceversion | 1  | varchar(24)      
devicezoneuri | 1 | varchar(180)     
dvchost     | 1         | varchar(20)      
version     | 1         | varchar(20)      
ahost       | 1         | varchar(20)      
art         | 1         | varchar(26)      
at          | 1         | varchar(22)      
cat         | 1         | varchar(28)      
catdt       | 1         | varchar(36)      
devicevendor | 1  | varchar(20)      
dtz         | 1         | varchar(32)      
dvc         | 1         | varchar(24)      
```
3. Query several columns from the logs_view:

```sql
=> \x
Expanded display is on.
VMart=> select version, devicevendor, deviceversion, name, severity, signatureid
    from logs_view;
-[ RECORD 1 ]-+-------------------------------------+
  version | 0
  devicevendor | ArcSight
  deviceversion | 6.0.3.6664.0
  name | Agent [test] type [testalerting] started
  severity | Low
  signatureid | agent:030
```

Use the fcefparser Delimiter Parameter

In this example, you use the `fcefparser` delimiter parameter to query events located in California, New Mexico, and Arizona.

1. Create a new columnar table, CEFData3:

```sql
=> CREATE TABLE CEFData3(eventId INT, location VARCHAR(20));
CREATE TABLE
```

2. Using the `delimiter=', '` parameter, load some CEF data into the table:

```sql
=> COPY CEFData3 FROM stdin PARSER fcefparser(delimiter=', ');
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.
```
Loading CSV Data

Use the `fcsvparser` to load data in CSV format (comma-separated values). Since no formal CSV standard exists, Vertica supports the RFC 4180 standard as the default behavior for `fcsvparser`. Other parser parameters simplify various combinations of CSV options into columnar or flex tables. Using `fcsvparser` parses the following CSV data formats:

- **RFC 4180**: The RFC4180 CSV format parser for Vertica flex tables. The parameters for this format are fixed and cannot be changed.

- **Traditional**: The traditional CSV parser lets you specify the parameter values such as delimiter or record terminator. For a detailed list of parameters, please refer to the `FCSVPARSER` page.

Using Default Parser Settings

These fixed parameter settings apply to the RCF4180 format.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Fixed Value (RCF4180)</th>
<th>Default Value (Traditional)</th>
</tr>
</thead>
</table>

Note: While you can change the default parser parameter values for Traditional format files, each value must be unique. For example, you can specify an ampersand (&) as a delimiter. If you do, you cannot also use an ampersand for the escape or other parameter value.
Use the `type` parameter to indicate either an RFC 4180-compliant file or a traditional-compliant file. You can specify `type` as `RCF4180`. However, you must first verify that the data is compatible with the preceding fixed values for parameters of the RFC4180 format. The default value of the `type` parameter is `RFC4180`.

### Loading CSV Data (RFC4180)

Follow these steps to use `fcsvparser` to load data in the RFC4180 CSV data format.

To perform this task, assume that the CSV file to be loaded has the following sample contents:

```
$ more /home/dbadmin/flex/flexData1.csv
sno,name,age,gender
1,John,14,male
2,Mary,23,female
3,Mark,35,male
```

1. **Create a flex table:**

   ```sql
   => CREATE FLEX TABLE csv_basic();
   CREATE TABLE
   ```

2. **Load the data from the CSV file using `fcsvparser`:**

   ```sql
   => COPY csv_basic FROM '/home/dbadmin/flex/flexData1.csv' PARSER fcsvparser();
   Rows Loaded
   -------------------
   3 (1 row)
   ```

3. **View the data loaded in the flex table:**

   ```sql
   => SELECT maptoString(_raw_) FROM csv_basic;
   maptoString
   ---------------------------------------------------------------
   {
   "age" : "14",
   "gender" : "male",
   "name" : "John",
   ```
"sno" : "1"
}
{
  "age" : "23",
  "gender" : "female",
  "name" : "Mary",
  "sno" : "2"
}
{
  "age" : "35",
  "gender" : "male",
  "name" : "Mark",
  "sno" : "3"
}
(3 rows)

### Loading CSV Data (Traditional)

Follow these steps to use `fcsvparser` to load data in traditional CSV data format using `fcsvparser`.

In this example, the CSV file uses `$` as a delimiter and `#` as a record_terminator. The sample CSV file to load has the following contents:

```
$ more /home/dbadmin/flex/flexData1.csv
sno$name$age$gender#
1$John$14$male#
2$Mary$23$female#
3$Mark$35$male#
```

1. Create a flex table:

```sql
=> CREATE FLEX TABLE csv_basic();
CREATE TABLE
```

2. Load the data in flex table using `fcsvparser` with parameters type='traditional', delimiter='$' and record_terminator='#':

```sql
=> COPY csv_basic FROM '/home/dbadmin/flex/flexData2.csv' PARSER fcsvparser(type='traditional', delimiter='$', record_terminator='#');
Rows Loaded
----------
3
(1 row)
```

3. View the data loaded in the flex table:

```sql
=> SELECT maptostring(__raw__) FROM csv_basic;
maptostring
```
### Rejecting Duplicate Values

You can reject duplicate values using the `reject_on_duplicate=true` option with the `fcsvparser`. The load continues after it rejects a duplicate value. The next example shows how to use this parameter and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```sql
=> CREATE FLEX TABLE csv_basic();
CREATE TABLE

=> COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_duplicate=true)
exceptions '/home/dbadmin/load_errors/except.out' rejected data '/home/dbadmin/load_errors/reject.out';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

> | |
> 1 | 2
> \.

=> `! cat /home/dbadmin/load_errors/reject.out
A|A
=> `! cat /home/dbadmin/load_errors/except.out
A|A
COPY: Input record 1 has been rejected (Processed a header row with duplicate keys with reject_on_duplicate specified; rejecting.). Please see /home/dbadmin/load_errors/reject.out, record 1 for the rejected record.
COPY: Loaded 0 rows, rejected 1 rows.
```
Rejecting Data on Materialized Column Type Errors

The fcsvparser parser has a Boolean parameter, `reject_on_materialized_type_error`. Setting this parameter to true causes rows to be rejected if both the following conditions exist in the input data:

- Includes keys matching an existing materialized column
- Has a key value that cannot be coerced into the materialized column's data type

The following examples illustrate setting this parameter.

1. Create a table, `reject_true_false`, with two real columns:

   ```sql
   => CREATE FLEX TABLE reject_true_false(one int, two int);
   CREATE TABLE
   ```

2. Load CSV data into the table (from STDIN), using the fcsvparser with `reject_on_materialized_type_error=false`. While false is the default value, you can specify it explicitly, as shown. Additionally, set the parameter `header=true` to specify the columns for input values:

   ```sql
   => COPY reject_true_false FROM stdin PARSER fcsvparser(reject_on_materialized_type_error=false,header=true);
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> one,two
   >> 1,2
   >> "3","four"
   >> "five",6
   >> 7,8
   >> 
   ```

3. Invoke `maptostring` to display the table values after loading data:

   ```sql
   => SELECT maptostring(_raw_), one, two FROM reject_true_false;
   maptostring | one | two
   ---------------------------------+-----+-----
   {             "one" : "1",
   "two" : "2"
   }         1   2
   {             "one" : "3",
   "two" : "four"
   }
   ```
4. Truncate the table to empty the data stored in the table:

```sql
=> TRUNCATE TABLE reject_true_false;
TRUNCATE TABLE
```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

```sql
=> COPY reject_true_false FROM stdin PARSER fcsvparser(reject_on_materialized_type_error=true, header=true);
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> one,two
>> 1,2
>> "3","four"
>> "five",6
>> 7,8
>> .
```

6. Call `maptostring` to display the table contents. Only two rows are currently loaded, whereas the previous results had four rows. The rows having input values with incorrect data type have been rejected:

```sql
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;
maptostring | one | two
-------------|-----|-----
{ "one" : "1", "two" : "2" } | 1 | 2
{ "one" : "7", "two" : "8" } | 7 | 8
(2 rows)
```
Note: The parser fcsvparser uses null values if there is a type mismatch and you set the reject_on_materialized_type_error parameter to false.

Rejecting or Omitting Empty Rows

Valid CSV files can include empty key and value pairs. Such rows are invalid for SQL. You can control the behavior for empty rows by either rejecting or omitting them, using two boolean FCSVPARSER parameters:

- reject_on_empty_key
- omit_empty_keys

The following example illustrates how to set these parameters:

1. Create a flex table:

   ```
   => CREATE FLEX TABLE csv_basic();
   CREATE TABLE
   ```

2. Load CSV data into the table (from STDIN), using the fcsvparser with reject_on_empty_key=false. While false is the default value, you can specify it explicitly, as shown. Additionally, set the parameter header=true to specify the columns for input values:

   ```
   => COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_empty_key=false,header=true);
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> ,num
   >> 1,2
   >> 
   ```

3. Invoke maptostring to display the table values after loading data:

   ```
   =>SELECT maptostring(_raw_) FROM csv_basic;
   maptostring
   --------------------------
   {"" : "1",
   "num" : "2"
   }
   (1 row)
   ```
4. Truncate the table to empty the data stored in the table:

```sql
=> TRUNCATE TABLE csv_basic;
TRUNCATE TABLE
```

5. Reload the same data again, but this time, set reject_on_empty_key=true:

```sql
=> COPY csv_basic FROM stdin PARSER fcsvparser(reject_on_empty_key=true,header=true);
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> ,num
>> 1,2
>> \. 
```

6. Call maptostring to display the table contents. No rows are loaded because one of the keys is empty:

```sql
=>SELECT maptostring(_raw_) FROM csv_basic;
maptostring
-----------------------
(0 rows)
```

7. Truncate the table to empty the data stored in the table:

```sql
=> TRUNCATE TABLE csv_basic;
TRUNCATE TABLE
```

8. Reload the same data again, but this time, set omit_empty_keys=true:

```sql
=> COPY csv_basic FROM stdin PARSER fcsvparser(omit_empty_keys=true,header=true);
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> ,num
>> 1,2
>> \. 
```

9. Call maptostring to display the table contents. One row is now loaded, and the rows with empty keys are omitted:

```sql
=> SELECT maptostring(_raw_) FROM csv_basic;
maptostring
-----------------------
{
  "num" : "2"
}
(1 row)
```

Note: If no header names exist, `fcsvparser` uses a default header of `ucoln`, where `n` is the column offset number. If a table header name and key name match, the parser loads
the column with values associated with the matching key name.

Using the NULL Parameter

Use the COPY NULL metadata parameter with fcsvparser to load NULL values into a flex table.

The next example uses this parameter:

1. Create a flex table.

```sql
=> CREATE FLEX TABLE fcsv(c1 int);
CREATE TABLE
```

2. Load CSV data in flex table using STDIN and NULL parameter.

```sql
=> COPY fcsv FROM STDIN PARSER fcsvparser() NULL 'NULL';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
   >> a,b,c1
   >> 10,20,NULL
   >> 20,30,50
   >> 20,30,40
   >> .
```

3. Use compute_flextable_keys_and_build_view function to compute keys and build flex view.

```sql
=> SELECT compute_flextable_keys_and_build_view('fcsv');
compute_flextable_keys_and_build_view
---------------------------------------------------------------------------------------------------------------------
Please see public.fcsv_keys for updated keys
The view public.fcsv_view is ready for querying
(1 row)
```

4. View the flex view and replace the NULL values.

```sql
=> SELECT * FROM public.fcsv_view;
a | b | c1
----------
20 | 30 | 50
10 | 20 |
20 | 30 | 40
(3 rows)
=> SELECT a,b, ISNULL(c1,-1) from public.fcsv_view;
a | b | ISNULL
----------
20 | 30 | 50
10 | 20 | -1
```
Handling Column Headings

The fcsvparser lets you specify your own column headings with the HEADER_NAMES= parameter. This parameter entirely replaces column names in the CSV source header row.

For example, to use these six column headings for a CSV file you are loading, use the fcsvparser parameter as follows:

```
HEADER_NAMES='FIRST, LAST, SOCIAL_SECURITY, TOWN, STATE, COUNTRY'
```

Supplying fewer header names than existing data columns causes fcsvparser to use default names after those you supply. Default header names consist of ucoln, where n is the column offset number, starting at 0 for the first column. For example, if you supply four header names for a 6-column table, fcsvparser supplies the default names ucol4 and ucol5, following the fourth header name you provide.

If you supply more headings than the existing table columns, any additional headings remain unused.

Loading Delimited Data

You can load flex tables with one of two delimited parsers, fdelimitedparser or fdelimitedpairparser.

- Use fdelimitedpairparser when the data specifies column names with the data in each row.
Use `fdelimitedparser` when the data does not specify column names or has a header row for column names.

This section describes using some options that `fdelimitedpairparser` and `fdelimitedparser` support.

### Rejecting Duplicate Values

You can reject duplicate values using the `reject_on_duplicate=true` option with the `fdelimitedparser`. The load continues after it rejects a duplicate value. The next example shows how to use this parameter and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```sql
=> CREATE FLEX TABLE delim_dupes();
CREATE TABLE
=> COPY delim_dupes FROM stdin PARSER fdelimitedparser(reject_on_duplicate=true)
exceptions '/home/dbadmin/load_errors/except.out' rejected data '/home/dbadmin/load_errors/reject.out';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> A|A
>> 1|2
>> .
=> \! cat /home/dbadmin/load_errors/reject.out
A|A
=> \! cat /home/dbadmin/load_errors/except.out
COPY: Input record 1 has been rejected (Processed a header row with duplicate keys with reject_on_duplicate specified; rejecting.). Please see /home/dbadmin/load_errors/reject.out, record 1 for the rejected record.
COPY: Loaded 0 rows, rejected 1 rows.
```

### Rejecting Materialized Column Type Errors

Both the `fjsonparser` and `fdelimitedparser` parsers have a boolean parameter, `reject_on_materialized_type_error`. Setting this parameter to `true` causes rows to be rejected if both the following conditions exist in the input data:

- Includes keys matching an existing materialized column
- Has a value that cannot be coerced into the materialized column's data type

Suppose the flex table has a materialized column, `OwnerPercent`, declared as a `FLOAT`. Trying to load a row with an `OwnerPercent` key that has a `VARCHAR` value causes `fdelimitedparser` to reject the data row.
The following examples illustrate setting this parameter.

1. Create a table, `reject_true_false`, with two real columns:

   ```sql
   CREATE FLEX TABLE reject_true_false(one VARCHAR, two INT);
   ```

2. Load JSON data into the table (from STDIN), using the `fjsonparser` with `reject_on_materialized_type_error=false`. While false is the default value, the following example specifies it explicitly for illustration:

   ```sql
   COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=false);
   ```

3. Invoke `maptostring` to display the table values after loading data:

   ```sql
   SELECT maptostring(_raw_), one, two FROM reject_true_false;
   ```

4. Truncate the table:

   ```sql
   TRUNCATE TABLE reject_true_false;
   ```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

   ```sql
   COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=true);
   ```
End with a backslash and a period on a line by itself.

```plaintext
>> {"one": 1, "two": 2}
>> {"one": "one", "two": "two"}
>> {"one": "one", "two": 2}
>> \
```

6. Call `maptoString` to display the table contents. Only two rows were loaded, whereas the previous results had three rows:

```sql
=> SELECT maptoString(__raw__), one, two FROM reject_true_false;
maptoString  | one | two
-------------|-----|-----
{             |
    "one": "1",
    "two": "2"
}            | 1   | 2
{            |
    "one": "one",
    "two": "2"
}            | one | 2
(2 rows)
```

## Loading JSON Data

You can load JSON data into flex or columnar tables. This section describes some examples of using the `fjsonparser` with several of the parser options.

### Checking JSON Integrity

Before loading any JSON data, be sure that the data is valid. You can verify JSON data integrity using a web tool such as [JSONLint](https://jsonlint.com). Copy your JSON data into the tool. If any data is invalid, the tool returns a message similar to the one in this example:

```plaintext
Parse error on line 170:...257914002502451200}
    "id_str": "257
------------------------^
Expecting 'EOF', '}', ',', ']'
```
Using flatten_maps and flatten_arrays Parameters

When loading JSON data, the fjsonparser uses the parameters flatten_maps and flatten_arrays to control how the parser handles the data it is loading. Here are the default settings for these two parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Change Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_maps</td>
<td>TRUE: Flatten all maps.</td>
<td>flatten_maps=FALSE</td>
</tr>
<tr>
<td>flatten_arrays</td>
<td>FALSE: Do not flatten arrays.</td>
<td>flatten_arrays=TRUE</td>
</tr>
</tbody>
</table>

You control the default the behavior by using one or both flatten_ parameters.

For JSON maps, the parser flattens all submaps, separating the levels with a period (.).
Consider the following input data with a submap:

```json
{ grade: { level: 4 } }
```

The default parser behavior results in the following map:

```json
{ "grade.level" -> "4" }
```

**Note:** To use the bracket operators ([]) to access deeply nested JSON in VMap data, you must load the data with flatten_maps=FALSE, as described in Querying Nested Data.

For JSON arrays, the parser maintains the array. Consider the following input data containing a 2-element array, with values 1 and 2:

```json
{ grade: [ 1 2 ] }
```

The default parser behavior results in the following array:

```json
{ "grade": { "0" -> "1", "1" -> "2" } }
```

**Note:** Using the parameters flatten_maps and flatten_arrays is recursive, and affects all data.
Loading from a Specific Start Point

You can use the `fjsonparser start_point` parameter to load JSON data beginning at a specific key, rather than at the beginning of a file. Data is parsed from after the `start_point` key until the end of the file, or to the end of the first `start_point`'s value. The `fjsonparser` ignores any subsequent instance of the `start_point`, even if that key appears multiple times in the input file. If the input data contains only one copy of the `start_point` key, and that value is a list of JSON elements, the parser loads each element in the list as a row.

This section uses the following sample JSON data, saved to a file (`alphanums.json`):

```json
{   "A": {   "B": {   "C": [ {   "d": 1,   "e": 2,   "f": 3 }, {   "g": 4,   "h": 5,   "i": 6 },
   {   "j": 7,   "k": 8,   "l": 9 } ] } }
```

1. Create a flex table, `start_json`:

```sql
=> CREATE FLEX TABLE start_json();
CREATE TABLE
```

2. Load `alphanums.json` into `start_json` using the `fjsonparser` without any parameters:

```sql
=> COPY start_json FROM '/home/dbadmin/data/flex/alphanums.json' PARSER fjsonparser();
Rows Loaded
-------------------
 1
(1 row)
```

3. Use `maptostring` to see the results of loading all of `alphanums.json`:

```sql
=> SELECT maptostring(_raw_) FROM start_json;
maptostring
----------------------------------------
{   "A.B.C" : {
   "0.d" : "1",
   "0.e" : "2",
   "0.f" : "3",
   "1.g" : "4",
   "1.h" : "5",
   "1.i" : "6",
   "2.j" : "7",
   "2.k" : "8",
   "2.l" : "9"
}
}
4. Truncate start_json and load alphanums.json with the start_point parameter:

```
=> TRUNCATE TABLE start_json;
TRUNCATE TABLE
=> COPY start_json FROM '/home/dbadmin/data/flex/alphanums.json' PARSE
-> fjsonparser(start_point='B');
Rows Loaded
--------------
1
(1 row)
```

5. Next, call maptostring again to compare the results of loading alphanums.json from start_point='B':

```
=> SELECT maptostring(_raw_) FROM start_json;
maptostring
--------------
{
  "C" : {
    "0.d" : "1",
    "0.e" : "2",
    "0.f" : "3",
    "1.g" : "4",
    "1.h" : "5",
    "1.i" : "6",
    "2.j" : "7",
    "2.k" : "8",
    "2.l" : "9"
  }
}
(1 row)
```

### Parsing From a Start Point Occurrence

If a start_point value occurs in multiple locations in your JSON data, you can use the start_point_occurrence integer parameter to specify the occurrence at which to start parsing. By defining start_point_occurrence, fjsonparser begins at the nth occurrence of start_point.

### Controlling Column Name Separators

By default, fjsonparser produces column names by concatenating JSON field names with a period (.). You can change the default separator by specifying a different character with the
key_separator parameter.

**Handling Special Characters**

Some input JSON data can have special characters in field names. You can replace these characters by setting the `suppress_nonalphanumeric_key_chars` to `TRUE`. With this parameter setting, all special characters are converted to an underscore `_` character.

**Dealing with Invalid JSON Records**

If your JSON data is not perfectly formatted, your load may fail due to invalid records. You can use the `RECORD_TERMINATOR` parameter to skip these invalid records if your JSON records are consistently delimited by a character like a line break. Setting a record terminator will allow the FJSONPARSER to skip over invalid records and continue parsing the rest of the data.

If your records are not consistently marked by a character, you can use the `COPY` parameter `ERROR TOLERANCE`. `ERROR TOLERANCE` skips entire source files with invalid JSON records, while `RECORD_TERMINATOR` skips individual malformed JSON records. Using both `ERROR TOLERANCE` and `RECORD_TERMINATOR` within the statement will work, but if your records are consistently marked, `RECORD_TERMINATOR` should sufficiently deal with imperfect records.

1. Create a flex table named "fruits".

   ```sql
   => CREATE FLEX TABLE fruits();
   CREATE TABLE
   ```

2. Use the FJSONPARSER and call the `RECORD_TERMINATOR` parameter with a termination key of `E'\n'` (which denotes a new line). Insert records, including an invalid record.

   ```sql
   => COPY fruits FROM STDIN PARSER FJSONPARSER(RECORD_TERMINATOR=E'\n');
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself
   >>{"name": "orange", "type": "fruit", "color": "orange", "rating": 5 }
   >>{"name": "apple", "type": "fruit", "color": "green" }
   >>{"name": "blueberry", "type": "fruit", "color": "blue", "rating": 10 }
   >>"type": "fruit", "rating": 7 }
   >>{"name": "banana", "type": "fruit", "color": "yellow", "rating": 3 }
   >>.  
   ```

3. View the flex table using MAPTOSTRING to confirm that the invalid record was skipped while the rest of the records were successfully loaded.

   ```sql
   => SELECT MAPTOSTRING(__raw__) FROM fruits;
   maptostring
   ```
Rejecting Duplicate Values

You can reject duplicate values by using the `reject_on_duplicate=true` option with the `fjsonparser`. The next example uses this option while loading data and then displays the specified exception and rejected data files. Saving rejected data to a table, rather than a file, includes both the data and its exception.

```sql
=> CREATE FLEX TABLE json_dupes();
CREATE TABLE

=> COPY json_dupes FROM stdin PARSER fjsonparser(reject_on_duplicate=true) exceptions '/home/dbadmin/load_errors/json_e.out' rejected data '/home/dbadmin/load_errors/json_r.out';
COPY: Input record 1 has been rejected (Rejected by user-defined parser).
Please see /home/dbadmin/load_errors/json_r.out, record 1 for the rejected record.
COPY: Loaded 0 rows, rejected 1 rows.
```
Rejecting Data on Materialized Column Type Errors

Both the fjsonparser and fdelimitedparser parsers have a Boolean parameter, reject_on_materialized_type_error. Setting this parameter to true causes rows to be rejected if the input data:

- Includes keys matching an existing materialized column
- Has a key value that cannot be coerced into the materialized column's data type.

The following examples illustrate setting this parameter.

1. Create a table, reject_true_false, with two real columns:

```sql
=> CREATE FLEX TABLE reject_true_false(one VARCHAR, two INT);
CREATE TABLE
```

2. Load JSON data into the table (from STDIN), using the fjsonparser with reject_on_materialized_type_error=false. While false is the default value, the following example specifies it explicitly for illustration:

```sql
=> COPY reject_true_false FROM stdin PARSER fjsonparser(reject_on_materialized_type_error=false);
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> {"one": 1, "two": 2}
=> {"one": "one", "two": "two"}
=> {"one": "one", "two": 2}
=> .
```

3. Invoke maptostring to display the table values after loading data:

```sql
=> SELECT maptostring(_raw_), one, two FROM reject_true_false;
maptostring | one | two
-------------+-----+-----
{           |     |     
  "one" : "one",
  "two" : "2"
} | one | 2
{           |     |     
  "one" : "1",
  "two" : "2"
} | 1   | 2
{           |     |     
  "one" : "one",
  "two" : "two"
} |     |     
```
4. Truncate the table:

```
=> TRUNCATE TABLE reject_true_false;
```

5. Reload the same data again, but this time, set `reject_on_materialized_type_error=true`:

```
=> COPY reject_true_false FROM stdin
PARSER fjsonparser(reject_on_materialized_type_error=true);
```

Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

```
>> {"one": 1, "two": 2}
>> {"one": "one", "two": "two"}
>> {"one": "one", "two": 2}
>> 
```

6. Call `maptostring` to display the table contents. Only two rows were loaded, whereas the previous results had three rows:

```
=> SELECT maptostring(__raw__), one, two FROM reject_true_false;
```

```
+---------------------+-----+
| maptostring         | one | two |
|---------------------+-----+-----|
| { "one": "1",      |     |     |
|  "two": "2"        |     |     |
| }                   | 1   | 2   |
| { "one": "one",    |     |     |
|  "two": "2"        |     |     |
| }                   | one | 2   |
| (2 rows)            |     |     |
```

### Rejecting or Omitting Empty Rows

Valid JSON files OpenText can include empty key and value pairs, such as this one:

```
{"": 1 "}
```

Such rows are invalid for SQL. To prevent this situation, you can control the behavior for empty rows, either rejecting or omitting them. You do so using two boolean parameters for the parsers `FDELIMITEDPARSER` or `FJSONPARSER:`
- `reject_on_empty_key`
- `omit_empty_keys`

See Also
- Manually Copying Data From Kafka

## Loading Matches from Regular Expressions

You can load flex or columnar tables with the matched results of a regular expression, using the `fregexparsr`. This section describes some examples of using the options that the flex parsers support.

### Sample Regular Expression

These examples use the following regular expression, which searches information that includes the `timestamp`, date, `thread_name`, and `thread_id` strings.

```
Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.
```

This example expression loads any `thread_id` hex value, regardless of whether it has a `0x` prefix, `<thread_id>(?:0x)?[0-9a-f]+`.

```
'^(?<time>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d+)(?<thread_name>[A-Za-z ]*):(?<thread_id>\d*?)(?:0x)?[0-9a-f]+)-?<(?<transaction_id>[0-9a-f]+)?(?:[<component>\w+]]\d+)<(?<level>[\w\d]+)>(?<elevel>[\w\d]+)@[<?(<node>[\w\d]+)?:  ](?;<text>.*')
```

### Using Regular Expression Matches for a Flex Table

You can load the results from a regular expression into a flex table, using the `fregexparsr`. For a complete example of doing so, see [FREGEXPARSER](#).
Using fregexparser for Columnar Tables

This section illustrates how to load the results of a regular expression used with a sample log file for a Vertica database. By using an external table definition, the section presents an example of using fregexparser to load data into a columnar table. Using a flex table parser for a columnar tables gives you the capability to mix data loads in one table. For example, you can load the results of a regular expression in one session, and JSON data in another.

The following basic examples illustrate this usage.

1. Create a columnar table, vlog, with the following columns:

```sql
=> CREATE TABLE vlog (  
   "text" varchar(2322),  
   thread_id varchar(28),  
   thread_name varchar(44),  
   "time" varchar(46),  
   component varchar(30),  
   level varchar(20),  
   transaction_id varchar(32),  
   elevel varchar(20),  
   enode varchar(34)
);```

2. Use COPY to load parts of a log file using the sample regular expression presented above, with the fregexparser. Be sure to remove any line characters from this expression example before trying it yourself:

```sql
=> COPY v_log FROM '/home/dbadmin/data/flex/vertica.log' PARSER FRegexParser(pattern='\^\(\?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d.\d+\)\n\(<\?thread_name>[A-Za-z ]+:\)?\<thread\id\:\(?0x\)?[0-9a-f]+\)\n-?\(?\<transaction\_id>[0-9a-f]+\)\)?\(?\<component\>[w+]?\)\n\<\(?\<level\>[w+]?\)\)?\(?\<elevel\>[w+]?\)\)?\(?\<enode\>[w+]?\)\)?\(?\<text\>[\^]*\)\)' rejected data as table fregex_reject;
```

3. Query the time column:

```sql
=> SELECT time FROM flogs limit 10;

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>2014-04-02 04:02:02.614</td>
</tr>
<tr>
<td>2014-04-02 04:02:51.008</td>
</tr>
<tr>
<td>2014-04-02 04:02:51.010</td>
</tr>
<tr>
<td>2014-04-02 04:02:51.012</td>
</tr>
<tr>
<td>2014-04-02 04:02:51.012</td>
</tr>
</tbody>
</table>
| 2014-04-02 04:02:51.013  |```
Using External Tables with fregexparser

By creating an external columnar table for your Vertica log file, querying the table will return updated log information. The following basic example illustrate this usage.

1. Create a columnar table, vertica_log, using the AS COPY clause and fregexparser to load matched results from the regular expression. For illustrative purposes, this regular expression has new line characters to split long text lines. Remove any line returns before testing with this expression:

```sql
=> CREATE EXTERNAL TABLE public.vertica_log
  (
    "text" varchar(2322),
    thread_id varchar(28),
    thread_name varchar(44),
    "time" varchar(46),
    component varchar(30),
    level varchar(20),
    transaction_id varchar(32),
    elevel varchar(20),
    enode varchar(34)
  )
AS COPY FROM '/home/dbadmin/data/vertica.log'
PARSER FRegexParser(pattern='^\d\d\d\d-\d\d-\d\d\d\d:\d\d:\d\d.\d+$'
  (?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d\d\d.\d+)
  (?<thread_name>[A-Za-z ]):(?<thread_id>\d\d\d\d):(?<component>[a-zA-Z0-9-]+)
  -(?:<transaction_id>[a-zA-Z0-9-]+)<(?<level>[a-zA-Z0-9-]+)>\n  (?:<elevel>[a-zA-Z0-9-]+)\n  @(?<enode>[a-zA-Z0-9-]+):\n  (?<text>.*)');
```

2. Query from the external table to get updated results:

```sql
=> SELECT component, thread_id, time FROM vertica_log limit 10;
```

<table>
<thead>
<tr>
<th>component</th>
<th>thread_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
<tr>
<td>Init</td>
<td>0x16321430</td>
<td>2014-04-02 04:02:02.613</td>
</tr>
</tbody>
</table>
| 0x16321430 | 2014-04-02 04:02:02.614 
(10 rows)
Computing Flex Table Keys

After loading data into a flex table, you can determine the set of keys that exist in the \_raw\_ column (the map data). Two helper functions compute keys from flex table map data:

- **COMPUTE_FLEXTABLE_KEYS**— Determines which keys exist as virtual columns in the flex map.

- **COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**— Performs the same functionality as COMPUTE_FLEXTABLE_KEYS, additionally building a new view. See also [Updating Flex Table Views](#).

Using COMPUTE_FLEXTABLE_KEYS

During execution, this function calculates the following information for the flex keys table columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_name</td>
<td>The name of the virtual column (key).</td>
</tr>
<tr>
<td>frequency</td>
<td>The number of times the key occurs in the map.</td>
</tr>
<tr>
<td>data_type_guess</td>
<td>The type guess that the helper functions determine for each distinct key in the map data. The function determines the type of each non-string value, depending on the length of the key, and whether the key includes nested maps. Changing the default value of the EnableBetterFlexTypeGuessing configuration parameter to 0 (OFF) results in the functions determining all flex table keys as string types ([LONG]VARCHAR) or ([LONG]VARBINARY).</td>
</tr>
</tbody>
</table>

Determining Key Data Types

By default, using COMPUTE_FLEXTABLE_KEYS determines non-string key values from the \_raw\_ column LONG VARBINARY type. The non-string keys include these data types (and others listed in [SQL Data Types](#)):
Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the EnableBetterFlexTypeGuessing configuration parameter. When the parameter is ON, the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as trees.csv).
2. Create a flex table (trees) and load trees.csv using the fcsvparser:

```sql
=> CREATE FLEX TABLE trees();
=> COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
```
3. Use COMPUTE_FLEXTABLE KEYS with the trees flex table.

```sql
=> SELECT COMPUTE_FLEXTABLE_KEY('trees');
COMPUTE_FLEXTABLE_KEY

Please see public.trees_keys for updated keys
(1 row)
```
4. Query the trees_keys table output:
5. Set the `EnableBetterFlexTypeGuessing` parameter to 0 (OFF).

6. Call `COMPUTE_FLEXTABLE_KEYS` with the `trees` flex table again.

7. Query the `trees_keys` table to compare the `data_type_guess` values with the previous results. Without the configuration parameter set, all of the non-string data types are VARCHARS of various lengths:

```
>> SELECT * FROM trees_keys;
key_name | frequency | data_type_guess
---------|-----------|------------------
Year     | 6         | Integer
Quarter  | 6         | Integer
Region   | 6         | Varchar(66)
Available| 6         | Boolean
Number of Quotes | 6       | Integer
Grade    | 6         | Varchar(20)
Species  | 6         | Varchar(22)
Pond Value| 6      | Varchar(20)
(8 rows)
```

8. To maintain accurate results for non-string data types, set the `EnableBetterFlexTypeGuessing` parameter back to 1 (ON).

For more information about setting the `EnableBetterFlexTypeGuessing` configuration parameter, see Setting Flex Table Configuration Parameters.

**Calculating Key Value Column Widths**

The `COMPUTE_FLEXTABLE_KEYS` function determines the column width for keys by determining the length of the largest value for each key, multiplied by the `FlexTableDataTypeGuessMultiplier` factor. For more about this configuration parameter, see Setting Flex Table Configuration Parameters.
The next example shows the results of populating the `_keys` table after creating a flex table (`darkdata1`) and loading data. The column widths are shown in parentheses, where applicable, after the value of the `data_type_guess` column:

```
=> SELECT compute_flextable_keys('darkdata1');
compute_flextable_keys

Please see public.dar1data1_keys for updated keys
(1 row)
```

```
=> SELECT * from darkdata1_keys;
key_name | frequency | data_type_guess
----------------------------------------
created_at | 8 | TimestampTz
delete.status.id_str | 4 | Integer
delete.status.user_id | 4 | Integer
entities.hashtags | 8 | long varbinary(186)
favorited | 8 | Boolean
id_str | 8 | Integer
in_reply_to_screen_name | 8 | Varchar(24)
retweeted.status.contributors | 1 | Varchar(20)
retweeted.status.coordinates | 1 | Varchar(20)
retweeted.status.created_at | 1 | TimestampTz
.
.
.(125 rows)
```
Materializing Flex Tables

Once flex tables exist, you can change the table structure to promote virtual columns to materialized (real) columns. If your table is already a hybrid table, you can change existing real columns and promote other important virtual columns. This section describes some key aspects of promoting columns, adding columns, specifying constraints, and declaring default values. It also presents some differences when loading flex or hybrid tables, compared with columnar tables.

Note: Materializing virtual columns by promoting them to real columns can significantly improve query performance. Vertica recommends that you materialize important virtual columns before running large and complex queries. Promoted columns cause a small decrease in load performance.

Adding Columns to Flex Tables

Add columns to your flex tables to promote virtual columns:

1. Add a real column with the same name as a virtual column (`user.name`):

   ```sql
   => ALTER TABLE darkdata1 ADD COLUMN "user.name" VARCHAR;
   ALTER TABLE
   ```

2. Load some data into the table.

   ```sql
   => COPY darkdata1 FROM '/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
   Rows Loaded
   12
   (1 row)
   ```

3. Query the materialized column. Notice that loading data populates the column automatically. Blank rows indicate no values or NULLs:

   ```sql
   => SELECT "user.name" FROM darkdata1;
   user.name
   ---------------------
   I'm Toaster@yahoo
   Flu Beach
   seydo shi
   ```
Adding Columns with Default Values

The section Bulk Loading Data into Flex Tables describes the use of default values, and how Vertica evaluates them during loading. As with all tables, using COPY to load data ignores any column default values.

Note: Adding a table column default expression to a flex table requires casting the column to an explicit data type.

1. Create a darkdata1 table with a column definition. The following example uses a column name (talker) that does not correspond to a virtual column name. Assign a default value with a virtual column name. In this example, the default value for the column talker is ("user.lang"). Since user.lang is a virtual column in the LONG VARBINARY _raw_ column, you must cast its value to VARCHAR to match the talker column definition:

   ```sql
   => CREATE FLEX TABLE darkdata1(talker VARCHAR default "user.lang"::VARCHAR);
   CREATE TABLE
   ```

2. Load some JSON data, specifying the _raw_ column:

   ```sql
   => COPY darkdata1 (_raw_) FROM '/test/vertica/flextable/DATA/tweets_12.json'
      PARSER fjsonparser();
   Rows Loaded
   ---------
   12
   (1 row)
   ```

3. Query the talker column. Notice that Vertica used the default column value ("user.lang"), because you specified _raw_. Blank rows indicate no values or NULLs:

   ```sql
   => SELECT "talker" FROM darkdata1;
   talker
   -------
   it
   ```
4. Alter the table to add a column with a known virtual column name (user.name), assigning the key name as the default value (recommended), and casting it to a VARCHAR:

```sql
=> ALTER TABLE darkdata1 ADD COLUMN "user.name" VARCHAR default "user.name"::VARCHAR;
```

5. Load data again, this time without __raw__:

```sql
=> COPY darkdata1 FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
```

6. Query the two real columns. Notice that talker has no values, because you did not specify the __raw__ column. The user.lang column contains values from the user.name virtual column:

```sql
=> SELECT "talker", "user.name" FROM darkdata1;
```

```plaintext
<table>
<thead>
<tr>
<th>talker</th>
<th>user.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>laughing at</td>
<td>Avita Desai</td>
</tr>
<tr>
<td>clouds.</td>
<td>I'm Toasterâ¥</td>
</tr>
<tr>
<td></td>
<td>Uptown gentle</td>
</tr>
<tr>
<td></td>
<td>man.</td>
</tr>
<tr>
<td>~G A B R I</td>
<td>Flu Beach</td>
</tr>
<tr>
<td>E L A â¿</td>
<td>seydo shi</td>
</tr>
<tr>
<td></td>
<td>The End</td>
</tr>
</tbody>
</table>
```

(12 rows)

7. Load data once more, this time specifying a COPY statement with a default value expression for user.name:

```sql
=> COPY darkdata1 (__raw__, "user.name" as 'QueenElizabeth'::varchar) FROM '/test/vertica/flextable/DATA/tweets_12.json' PARSER fjsonparser();
```

Rows Loaded
----------
12
8. Query once more. Notice that the real column talker has its default values (you used __raw__). As specified in COPY, the "user.name" as 'QueenElizabeth' expression overrode the user.name default column value:

```sql
=> SELECT "talker", "user.name" FROM darkdata1;
+----------+----------------+
| talker   | user.name     |
|----------+----------------|
| it       | QueenElizabeth |
| en       | QueenElizabeth |
| es       | QueenElizabeth |
| tr       | QueenElizabeth |
| en       | QueenElizabeth |
| en       | QueenElizabeth |
| es       | QueenElizabeth |
| en       | QueenElizabeth |
| tr       | QueenElizabeth |
| en       | QueenElizabeth |
(12 rows)
```

To summarize, you can set a default column value as part of the ALTER TABLE...ADD COLUMN... operation. For materializing columns, the default value should reference the key name of the virtual column (as in "user.lang"). Subsequently loading data with a COPY value expression overrides the default value of the column definition.

**Changing the __raw__ Column Size**

You can change the default size of the __raw__ column for flex tables you plan to create, the current size of an existing flex table, or both.

To change the default size for the flex table __raw__ column, use the following database configuration parameter (described in Setting Flex Table Configuration Parameters):

```sql
=> ALTER DATABASE mydb SET FlexTableRawSize = 120000;
```

Changing the configuration parameter affects all flex tables you create after making this change.

To change the size of the __raw__ column in an existing flex table, use the ALTER TABLE statement to change the definition of the __raw__ column:
Note: An error occurs if you try to reduce the __raw__ column size to a value smaller than any data the column contains.

**Changing Flex Table Real Columns**

You can make the following changes to the flex table real columns (__raw__ and __identity__), but not to any virtual columns:

<table>
<thead>
<tr>
<th>Actions</th>
<th><strong>raw</strong></th>
<th><strong>identity</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Change NOT NULL constraints (default)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Add primary key and foreign key (PK/FK) constraints</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Create projections</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Segment</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Partition</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Specify a user-defined scalar function (UDSF) as a default column expression in ALTER TABLE ( x ) ADD COLUMN ( y ) statement</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Note: While segmenting and partitioning the __raw__ column is permitted, it is not recommended due to its long data type. By default, if you not define any real columns, flex tables are segmented on the __identity__ column.

**Dropping Flex Table Columns**

There are two considerations about dropping columns:

- You cannot drop the last column in your flex table’s sort order.
- If you have not created a flex table with any real columns, or materialized any columns, you cannot drop the __identity__ column.
Updating Flex Table Views

Creating a flex table also creates a default view to accompany the table. The view has the name of the flex table with an underscore (_view) suffix. When you perform a select query from the default view, Vertica prompts you to run the helper function, `compute_flextable_keys_and_build_view`:

```sql
=> \dv dark*

<table>
<thead>
<tr>
<th>List of View Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

(2 rows)

=> SELECT * FROM darkdata_view;

Please run `compute_flextable_keys_and_build_view()` to update this view to reflect real and virtual columns in the flex table

(1 row)
```

There are two helper functions that create views:

- **COMPUTE_FLEXTABLE_KEYS** — See also **COMPUTE_FLEXTABLE_KEYS**
- **COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW** — Performs the same functionality as **BUILD_FLEXTABLE_KEYS** but also computes keys. See also Using **COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**.

Using **BUILD_FLEXTABLE_VIEW**

After computing keys for a flex table (**Computing Flex Table Keys**), call this function with one or more arguments. The records under the key_name column of the `{flextable}_keys` table are used as view columns, along with any values for the key. If no values exist, the column value is NULL.

Regardless of the number of arguments, calling this function replaces the contents of the existing view as follows:

<table>
<thead>
<tr>
<th>Function Invocation</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_flextable_view ('flexible_table')</td>
<td>Changes the existing view associated with flexible_</td>
</tr>
</tbody>
</table>
Function Invocation                  Results

<table>
<thead>
<tr>
<th></th>
<th>flexible_table_keystable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_flextable_view ('flexible_table', 'view_name')</td>
<td>Changes the view you specify with view_name by using the current contents of the {flextable}_keys table.</td>
</tr>
<tr>
<td>build_flextable_view ('flexible_table', 'view_name', 'table_keys')</td>
<td>Changes the view you specify with view_name to the current contents of the flexible_table_keys table. Use this function to change a view of your choice with the contents of the keys of interest.</td>
</tr>
</tbody>
</table>

If you do not specify a view_name argument, the default name is the flex table name with a _ view suffix. For example, if you specify the table darkdata as the sole argument to this function, the default view is called darkdata_view.

You cannot specify a custom view name with the same name as the default view flex_table_view, unless you first drop the default-named view and then create your own view of the same name.

Creating a view stores a definition of the column structure at the time of creation. Thus, if you create a flex table view and then promote virtual columns to real columns, you must rebuild the view. Querying a rebuilt flex table view that has newly promoted real columns produces two results. These results reflect values from both virtual columns in the map data and real columns.

**Handling JSON Duplicate Key Names in Views**

SQL is a case-insensitive language, so the names TEST, test, and TeSt are identical. JSON data is case sensitive, so that it can validly contain key names of different cases with separate values.

When you build a flex table view, the function generates a warning if it detects same-name keys with different cases in the {flextable}_keys table. For example, calling BUILD_FLEXTABLE_VIEW or COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW() on a flex table with duplicate key names results in these warnings:
While a `flextable` _keys_ table can include duplicate key names with different cases, a view cannot. Creating a flex table view with either of the helper functions consolidates any duplicate key names to one column name, consisting of all lowercase characters. All duplicate key values for that column are saved. For example, if these key names exist in a flex table:

- `test`
- `Test`
- `tESt`

The view will include a virtual column `test` with values from the `test`, `Test`, and `tESt` keys.

Note: The examples in this section include added Return characters to reduce line lengths. The product output may differ.

For example, consider the following query, showing the duplicate `test` key names:

```sql
=> \x
Expanded display is on.
dbt=> select * from dupe_keys;
- [ RECORD 1 ]----------------------------------------
   key_name | Test
   frequency | 2
   data_type_guess | varchar(20)
- [ RECORD 2 ]----------------------------------------
   key_name | Test
   frequency | 2
   data_type_guess | varchar(20)
- [ RECORD 3 ]----------------------------------------
   key_name | test
   frequency | 8
   data_type_guess | varchar(20)
- [ RECORD 4 ]----------------------------------------
```
Creating a Flex Table View

The following example shows how to create a view, `dd_view`, from the flex table `darkdata`, which contains JSON data.

```sql
=> CREATE VIEW dd_view AS SELECT "user.lang"::VARCHAR, "user.name"::VARCHAR FROM darkdata;
CREATE VIEW
```

Query the key names you specified, and their values:

```sql
=> SELECT * FROM dd_view;
user.lang | user.name
-----------|-------------
```
This example shows how to call `build_flextable_view` with the original table and the view you previously created, `dd_view`:

```sql
=> SELECT build_flextable_view ('darkdata', 'dd_view');
build_flextable_view
----------------------------------------
The view public.dd_view is ready for querying
(1 row)
```

Query the view again. You can see that the function populated the view with the contents of the `darkdata_keys` table. Next, review a snippet from the results, with the `key_name` columns and their values:

```sql
=> \x
Expanded display is on.

=> SELECT * FROM dd_view;
.
.
user.following |  
user.friends_count | 791  
user.geo_enabled | F  
user.id | 164464985  
user.id_str | 164464985  
user.is_translator | F  
user.lang | en  
user.listed_count | 4  
user.location | Uptown..  
user.name | Uptown gentleman.
.
.
```

When building views, be aware that creating a view stores a definition of the column structure at the time the view is created. If you promote virtual columns to real columns after building a view, the existing view definition is not changed. Querying this view with a select statement such as the following, returns values from only the `__raw__` column:
Also understand that rebuilding the view after promoting virtual columns changes the resulting value. Future queries return values from both virtual columns in the map data and from real columns.

### Using COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW

Call this function with a flex table to compute Flex table keys (see [Computing Flex Table Keys](#)), and create a view in one step.
Querying Flex Tables

After you create your flex table (with or without additional columns) and load data, you can perform these types of queries:

- SELECT
- COPY
- TRUNCATE
- DELETE

You can use SELECT queries for virtual columns that exist in the __raw__ column and other real columns in your flex tables. Column names are case insensitive.

Unsupported DDL and DML Statements

You cannot use the following DDL and DML statements with flex tables:

- CREATE TABLE flex_table AS...
- CREATE TABLE flex_table LIKE...
- SELECT INTO
- UPDATE
- MERGE

Determining Flex Table Data Contents

If you do not know what your flex table contains, two helper functions explore the VMap data to determine its contents. Use these functions to compute the keys in the flex table __raw__ column and, optionally, build a view based on those keys:
**COMPUTE_FLEXTABLE_KEYS**

**COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**

For more information about these and other helper functions, see [Flex Data Functions Reference](#).

To determine what key value pairs exist as virtual columns:

1. Call the function as follows:

```sql
=> SELECT compute_flextable_keys('darkdata');
compute_flextable_keys
-------------------------------------------
Please see public.darkdata_keys for updated keys (1 row)
```

2. View the key names by querying the `darkdata_keys` table:

```sql
=> SELECT * FROM darkdata_keys;
```

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>coordinates</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>created_at</td>
<td>8</td>
<td>varchar(60)</td>
</tr>
<tr>
<td>entities.hashtags</td>
<td>8</td>
<td>long varbinary(186)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>retweeted_status.user.time_zone</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>retweeted_status.user.url</td>
<td>1</td>
<td>varchar(68)</td>
</tr>
<tr>
<td>retweeted_status.user.utc_offset</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>retweeted_status.user.verified</td>
<td>1</td>
<td>varchar(20)</td>
</tr>
</tbody>
</table>

(125 rows)

### Querying Virtual Columns

Continuing with the JSON data example, use a SELECT statement query to explore content from the virtual columns. Then, analyze what is most important to you in case you want to materialize any virtual columns. This example shows how to query some common virtual columns in the VMap data:

```sql
=> SELECT "user.name", "user.lang", "user.geo_enabled" FROM darkdata1;
user.name      | user.lang | user.geo_enabled
----------------|-----------|-------------------|
laughing at clouds. | it       | T
Avita Desai    | en        | F
```

Vertica Analytic Database (9.0.x)
Querying Flex Table Keys

If you reference an undefined column ('which_column') in a flex table query, Vertica converts the query to a call to the maplookup() function as follows:

```
MAPLOOKUP(__raw__, 'which_column')
```

The maplookup() function searches the VMap data for the requested key and returns the following information:

- String values associated with the key for a row.
- NULL if the key is not found.

For more information about handling NULL values, see `MAPCONTAINSKEY()`.

Using Functions and Casting in Flex Table Queries

You can cast the virtual columns as required and use functions in your SELECT statement queries. The next example uses a SELECT statement to query the `created_at` and `retweet_count` virtual columns, and to cast their values in the process:

```
=> SELECT "created_at"::TIMESTAMP, "retweet_count"::INT FROM darkdata1 ORDER BY 1 DESC;
```

<table>
<thead>
<tr>
<th>created_at</th>
<th>retweet_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-10-15 14:41:05</td>
<td>0</td>
</tr>
<tr>
<td>2012-10-15 14:41:05</td>
<td>0</td>
</tr>
<tr>
<td>2012-10-15 14:41:05</td>
<td>0</td>
</tr>
<tr>
<td>2012-10-15 14:41:05</td>
<td>0</td>
</tr>
<tr>
<td>2012-10-15 14:41:05</td>
<td>0</td>
</tr>
</tbody>
</table>
The following query uses the COUNT and AVG functions to determine the average length of text in different languages:

```sql
=> SELECT "user.lang", count (*), avg(length("text"))::int FROM darkdata1 GROUP BY 1 ORDER BY 2 DESC;
```

<table>
<thead>
<tr>
<th>user.lang</th>
<th>count</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>4</td>
<td>42</td>
</tr>
<tr>
<td>es</td>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>it</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>tr</td>
<td>1</td>
<td>16</td>
</tr>
</tbody>
</table>

(Cast rows)

**Casting Data Types in a Query**

The following query requests the values of the `created_at` virtual column, without casting to a specific data type:

```sql
=> SELECT "created_at" FROM darkdata1;
```

```
created_at
-------------------
Mon Oct 15 18:41:04 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
Mon Oct 15 18:41:05 +0000 2012
(12 rows)
```

The next example queries the same virtual column, casting `created_at` to a `TIMESTAMP`. Casting results in different output and the regional time:

```sql
=> SELECT "created_at"::TIMESTAMP FROM darkdata1 ORDER BY 1 DESC;
```

```
created_at
--------------
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
2012-10-15 14:41:05
```
Accessing an Epoch Key

The term EPOCH (all uppercase letters) is reserved in Vertica for internal use.

If your JSON data includes a virtual column called epoch, you can query it within your flex table. However, use the maplookup() function to do so.
Querying Nested Data

If you load JSON or Avro data with `flatten_arrays=FALSE` (the default), VMap data in the `_raw_` column can contain multiple nested structures. In fact, any VMap JSON or Avro data can contain nested structures. This section describes how best to query such data.

Query VMap Nested Values

To query a nested structure, you can use multiple `maplookup()` functions, one for each level. However, the most efficient method is to use bracket (`[]`) operators.

When parsing or extracting VMap data, the default behavior is to flatten data. Flattened VMap data concatenates key names into one long name, delimiting elements with either the default delimiter (.), or a user-defined delimiter character.

To use bracket operators for nested structures in your VMap data, the data must not be flattened. Further, you cannot use bracket operators on any existing, flattened VMap data.

To load or extract VMap data correctly, specify `flatten_maps=FALSE` for `fjsonparser`, `favroparser`, and the `mapjsonextractor()` function.

**Note:** While bracket operator values look similar to array element specifications, they are strings, not integers. You must enter each nested structure as a string, even if the value is an integer. For example, if the value is 2, specify it as `['2']`, not `[2]`.

Bracket Operators For Nested JSON

This example uses the following JSON data as an example of nested data. Save this data as `restaurant.json`:

```json
{
   "restaurant": {
      ":name": "Bob's pizzeria",
      "cuisine": "Italian",
      "location": {"city": "Cambridge", "zip": "02140"},
      "menu": [{"item": "cheese pizza", "price": "$8.25"},
                 {"item": "chicken pizza", "price": "$11.99"},
                 {"item": "spinach pizza", "price": "$10.50"}]
   }
}
```

Create a flex table, `rests`, and load it with the `restaurant.json` file:
After loading your data into a flex table, there are two ways to access nested data using brackets:

- Beginning with the `__raw__` column, followed by the character values in brackets
- Starting with the name of the top-most element, followed by the character values in brackets

Both methods are equally efficient. Here are examples of both:

```sql
=> SELECT __raw__['restaurant']['location']['city'] FROM rests;

Cambridge (1 row)
=> SELECT restaurant['location']['city'] from rests;

restaurant Cambridge (1 row)
```

## Bracket Operators for Twitter Data

This example shows how to extract some basic information from Twitter data.

After creating a flex table, `tweets`, and loading in some data, the flex table has a block of tweets. In the following SELECT statement, notice how to specify the `__raw__` column of table `tweets`, followed by the bracket operators to define the virtual columns of interest (`['delete']['status']['user_id']`). This query uses the `COUNT()` function to calculate the number of deleted tweets and outputs 10 results:

```sql
=> SELECT __raw__['delete']['status']['user_id'] as UserId, COUNT(*) as TweetsDelete from tweets
   -> WHERE mapcontainskey(__raw__, 'delete')
   -> GROUP BY __raw__['delete']['status']['user_id']
   -> ORDER BY TweetsDelete DESC, UserId ASC LIMIT 10;

UserId | TweetsDelete
-------|--------------
106979547 | 4
483474369 | 4
181188657 | 3
```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>223136123</td>
<td>3</td>
</tr>
<tr>
<td>770139481</td>
<td>3</td>
</tr>
<tr>
<td>154602299</td>
<td>2</td>
</tr>
<tr>
<td>192127653</td>
<td>2</td>
</tr>
<tr>
<td>215011332</td>
<td>2</td>
</tr>
<tr>
<td>23321883</td>
<td>2</td>
</tr>
<tr>
<td>242173898</td>
<td>2</td>
</tr>
</tbody>
</table>

(10 rows)
Querying Flex Views

Flex tables offer the ability of dynamic schema through the application of query rewriting. Use flex views to support restricted access to flex tables. As with flex tables, each time you use a `SELECT` query on a flex table view, internally, Vertica invokes the `maplookup()` function, to return information on all virtual columns. This query behavior occurs for any flex or columnar table that includes a '__raw__' column.

This example illustrates querying a flex view:

1. Create a flex table.

```sql
=> CREATE FLEX TABLE twitter();
```

2. Load JSON data into flex table using `fjsonparser`.

```sql
=> COPY twitter FROM '/home/dbadmin/data/flex/tweets_10000.json' PARSER fjsonparser();
Rows Loaded
----------
10000
(1 row)
```

3. Create a flex view on top of flex table `twitter` with constraint `retweet_count>0`.

```sql
=> CREATE VIEW flex_view AS SELECT __raw__ FROM twitter WHERE retweet_count::int > 0;
CREATE VIEW
```

4. Query the view. First 5 rows are displayed.

```sql
=> SELECT retweeted,retweet_count,source FROM (select __raw__ from flex_view) t1 limit 5;
retweeted | retweet_count | source
-----------+---------------+---------------------------------------
F | 1 | <a href="http://blackberry.com/twitter" rel="nofollow">Twitter for BlackBerry®</a>
F | 1 | <a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>
F | 23 | <a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>
F | 7 | <a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>
(5 rows)
```
Listing Flex Tables

You can determine which tables in your database are flex tables by querying the `is_flextable` column of the `v_catalog.tables` system table. For example, use a query such as the following to see all tables with a true (t) value in the `is_flextable` column:

```sql
=> SELECT table_name, table_schema, is_flextable FROM v_catalog.tables;
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_schema</th>
<th>is_flextable</th>
</tr>
</thead>
<tbody>
<tr>
<td>bake1</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>bake1_keys</td>
<td>public</td>
<td>f</td>
</tr>
<tr>
<td>del</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>del_keys</td>
<td>public</td>
<td>f</td>
</tr>
<tr>
<td>delicious</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>delicious_keys</td>
<td>public</td>
<td>f</td>
</tr>
<tr>
<td>bake</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>bake_keys</td>
<td>public</td>
<td>f</td>
</tr>
<tr>
<td>appLog</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>appLog_keys</td>
<td>public</td>
<td>f</td>
</tr>
<tr>
<td>darkdata</td>
<td>public</td>
<td>t</td>
</tr>
<tr>
<td>darkdata_keys</td>
<td>public</td>
<td>f</td>
</tr>
</tbody>
</table>

(12 rows)
Setting Flex Table Configuration Parameters

These configuration parameters affect flex table usage:

- EnableBetterFlexTypeGuessing
- FlexTableRawSize
- FlexTableDataTypeGuessMultiplier

This section presents information about each of the parameters.

To change configuration parameters, and to understand parameter scope, see Setting Configuration Parameter Values and General Parameters in the Administrator's Guide.

Using EnableBetterFlexTypeGuessing

The EnableBetterFlexTypeGuessing configuration parameter is 1 (ON) by default. This setting results in the COMPUTE_FLEXTABLE_KEYS or COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW functions determining data types for string and non-string keys. If you change the configuration parameter to 0 (OFF), the functions determine all flex keys as string types ([LONG]VARCHAR) or ([LONG] VARBINARY).

You can set this configuration parameter at the database and session level.

For examples of using both settings, see Computing Flex Table Keys.

Specifying the FlexTableRawSize Parameter

The FlexTableRawSize parameter defines the default width of each flex table __raw__ column. This column is a LONG VARBINARY data type, and contains the map data you load into the table.

Setting this configuration parameter does not affect any existing flex tables, only the default width for new flex tables you create after changing FlexTableRawSize.

To change the __raw__ column width of an existing flex table, use the ALTER TABLE statement, described in Materializing Flex Tables.
Redefining the **FlexTableDataTypeGuessMultiplier**

After loading data into a flex table, each key in the `__raw__` column is a `LONG_VARBINARY` data type.

When you compute flex table keys with either of the functions, `COMPUTE_FLEXTABLE_KEYS` or `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`, the functions cast each key to the applicable Vertica data type. The key value length is determined by the key's largest value, multiplied by the `FlexTableDataTypeGuessMultiplier` factor. Padding the column width with the multiplier supports future data load key values at least twice the largest key value previously loaded.

The flex keys table that both flex functions populate is used to create the associated flex table view.

**Note:** The `FlexTableDataTypeGuessMultiplier` value is not used to calculate the width of any real columns in a flex table.
## Flex Data Functions Reference

The flex table data helper functions supply information you need to query the data you load. For example, suppose you don't know what keys are available in the map data. If not, you can use the `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW` function to populate a keys table and build a view. The functions aid in querying flex table and other VMap data.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>COMPUTE_FLEXTABLE_KEYS</code></td>
<td>Computes map keys from the map data in a flextable_data table, and populates the flextable_data_keys table with the computed keys. Use this function before building a view.</td>
</tr>
<tr>
<td><code>BUILD_FLEXTABLE_VIEW</code></td>
<td>Uses the keys in the flextable_data_keys table to create a view definition (flextable_data_view) for the flextable_data table. Use this function after computing flex table keys.</td>
</tr>
<tr>
<td><code>COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW</code></td>
<td>Performs both of the preceding functions in one call.</td>
</tr>
<tr>
<td><code>MATERIALIZE_FLEXTABLE_COLUMNS</code></td>
<td>Materializes a default number of columns (50) or more or less, if specified.</td>
</tr>
<tr>
<td><code>RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW</code></td>
<td>Replaces the flextable_data_keys table and the flextable_data_view, linking both the keys table and the view to the parent flex table.</td>
</tr>
</tbody>
</table>

While the functions are available to all users, they are applicable only to:

- Flex tables
- Associated `flex_table`_keys tables
- Associated `flex_table`_view views

By computing keys and creating views from flex table data, the functions allow you to perform `SELECT` queries. One function restores the original keys table and view that you specified when you first created the flex table.
Flex Table Dependencies

Each flex table (*flextable*) has two dependent objects:

- *flextable_keys*
- *flextable_view*

While both objects are dependent on their parent table, (*flextable*), you can drop either object independently. Dropping the parent table removes both dependents, without a CASCADE option.

Associating Flex Tables and Views

The helper functions automatically use the dependent table and view if they are internally linked with the parent table. You create both when you create the flex table. You can you drop either the _keys table or the _view, and re-create objects of the same name. However, if you do so, the new objects are not internally linked with the parent flex table.

In this case, you can restore the internal links of these objects to the parent table. To do so, drop the _keys table and the _view before calling the `RESTORE_FLEXTABLE_DEFAULT KEYS TABLE AND VIEW` function. Calling this function re-creates either, or both, the _keys table and the _view.

The remaining helper functions perform the tasks described in this section.

BUILD_FLEXTABLE_VIEW

Creates, or re-creates, a view for a default or user-defined _keys table, ignoring any empty keys.

Syntax

```
BUILD_FLEXTABLE_VIEW('[[database.]schema.]flex-table' [ [, 'view-name'] [ , 'user-keys-table'] ])
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>flex-table</td>
<td>The flex table name. By default, this function builds or rebuilds a view for the input table with the current contents of the associated flex_table_keys table.</td>
</tr>
<tr>
<td>view-name</td>
<td>A custom view name. Use this option to build a new view for flex-table with the name you specify.</td>
</tr>
<tr>
<td>user-keys-table</td>
<td>Specifies a keys table from which to create the view. Use this option if you created a custom user_keys table from the flex table map data, rather than from the default flex_table_keys table. The function builds a view from the keys in user_keys table, rather than from the flex_table_keys table.</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to call `build_flextable_view` with 1, 2, or 3 arguments.

Creating a Default View

To create, or re-create, a default view:

1. Call the function with an input flex table, darkdata:

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata');
build_flextable_view
--------------------------------------------
The view public.darkdata_view is ready for querying
(1 row)
```

The function creates a view with the default name (darkdata_view) from the darkdata_keys table.
2. Query a key name (user.id) from the new or updated view:

```sql
=> SELECT "user.id" FROM darkdata_view;
user.id
-----------------------
340857907
727774963
390498773
288187825
164464905
125434448
681328899
352494946
(12 rows)
```

Creating a Custom Name View

To create, or re-create, a view with a custom name:

1. Call the function with two arguments, an input flex table, darkdata, and the name of the view to create, dd_view:

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view');
build_flextable_view
------------------------------------------------------------------
The view public.dd_view is ready for querying
(1 row)
```

2. Query a key name (user.lang) from the new or updated view (dd_view):

```sql
=> SELECT "user.lang" FROM dd_view;
user.lang
------------
tr
en
es
en
en
it
es
en
(12 rows)
```

Creating a View from a Custom Keys Table

To create a view from a custom _keys table with build_flextable_view, the custom table must have the same schema and table definition as the default table (darkdata_keys).

Create a custom keys table, using any of these three approaches:
1. Create a columnar table with all keys from the default keys table for a flex table (darkdata_keys):

```sql
=> CREATE TABLE new_darkdata_keys AS SELECT * FROM darkdata_keys;
CREATE TABLE
```

2. Create a columnar table without content (LIMIT 0) from the default keys table for a flex table (darkdata_keys):

```sql
=> CREATE TABLE new_darkdata_keys AS SELECT * FROM darkdata_keys LIMIT 0;
CREATE TABLE
kd=> SELECT * FROM new_darkdata_keys;
key_name | frequency | data_type_guess
---------+-----------+---------------
(0 rows)
```

3. Create a columnar table without content (LIMIT 0) from the default keys table, and insert two values ('user.lang', 'user.name') into the key_name column:

```sql
=> CREATE TABLE dd_keys AS SELECT * FROM darkdata_keys limit 0;
CREATE TABLE
=> INSERT INTO dd_keys (key_name) values ('user.lang');
OUTPUT
-------
1
(1 row)
=> INSERT INTO dd_keys (key_name) values ('user.name');
OUTPUT
-------
1
(1 row)
=> SELECT * FROM dd_keys;
key_name | frequency | data_type_guess
---------+-----------+---------------
user.lang | 2         |               
user.name | 2         |               
(2 rows)
```

4. After creating a custom keys table, call build_flextable_view with all arguments (an input flex table, the new view name, the custom keys table):

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view', 'dd_keys');
build_flextable_view
---------------------------------------------------------------------
The view public.dd_view is ready for querying
(1 row)
```

5. Query the new view:
See Also

- `COMPUTE_FLEXTABLE.Keys`
- `COMPUTE_FLEXTABLE.Keys_AND_BUILD_VIEW`
- `MATERIALIZFLEXTABLE.COLUMNS`
- `RESTORE_FLEXTABLE_DEFAULT.Keys_TABLE_AND_VIEW`

**COMPUTE_FLEXTABLE.Keys**

Computes the virtual columns (keys and values) from the flex table VMap data. Use this function to compute keys without creating an associated table view. To also build a view, use `COMPUTE_FLEXTABLE.Keys_AND_BUILD_VIEW`.

The function stores its results in the associated flex_keys table, which has the following columns:

- `key_name`
- `frequency`
- `data_type_guess`

For more information, see Computing Flex Table Keys.

**Syntax**

```
COMPUTE_FLEXTABLE.Keys('[[database.]schema.]flex-table')
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For</td>
</tr>
</tbody>
</table>
Using Data Type Guessing

The results of the flex_keys table data_type_guess column depend on the EnableBetterFlexTypeGuessing configuration parameter. By default, the parameter is 1 (ON). This setting results in the function returning all non-string keys in the data_type_guess column as one of the following types (and others listed in SQL Data Types):

- BOOLEAN
- INTEGER
- FLOAT
- TIMESTAMP
- DATE

Setting the configuration parameter to 0 (OFF), results in the function returning only string types (LONGVARCHAR) or (LONGVARBINARY) for all values in the data_type_guess column of the flex_keys table.

Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the EnableBetterFlexTypeGuessing configuration parameter. When the parameter is ON, the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

<table>
<thead>
<tr>
<th>Year,Quarter,Region,Species,Grade,Pond Value,Number of Quotes,Available</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2015,1,2 - Northwest Oregon &amp; Willamette,Douglas-fir,1P,$615.12 ,12,No</td>
<td></td>
</tr>
<tr>
<td>2015,1,2 - Northwest Oregon &amp; Willamette,Douglas-fir,SM,$610.78 ,12,Yes</td>
<td></td>
</tr>
<tr>
<td>2015,1,2 - Northwest Oregon &amp; Willamette,Douglas-fir,2S,$596.00 ,20,Yes</td>
<td></td>
</tr>
<tr>
<td>2015,1,2 - Northwest Oregon &amp; Willamette,Hemlock,P,$520.00 ,6,Yes</td>
<td></td>
</tr>
<tr>
<td>2015,1,2 - Northwest Oregon &amp; Willamette,Hemlock,SM,$510.00 ,6,No</td>
<td></td>
</tr>
</tbody>
</table>
To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as trees.csv).

2. Create a flex table (trees) and load trees.csv using the fcsvparser:

   ```sql
   => CREATE FLEX TABLE trees();
   => COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
   ```

3. Use COMPUTE_FLEXTABLE.Keys with the trees flex table.

   ```sql
   => SELECT COMPUTE_FLEXTABLE.Keys('trees');
   ```

4. Query the trees_keys table output:

   ```sql
   => SELECT * FROM trees_keys;
   ```

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>6</td>
<td>Integer</td>
</tr>
<tr>
<td>Quarter</td>
<td>6</td>
<td>Integer</td>
</tr>
<tr>
<td>Region</td>
<td>6</td>
<td>Varchar(66)</td>
</tr>
<tr>
<td>Available</td>
<td>6</td>
<td>Boolean</td>
</tr>
<tr>
<td>Number of Quotes</td>
<td>6</td>
<td>Integer</td>
</tr>
<tr>
<td>Grade</td>
<td>6</td>
<td>Varchar(20)</td>
</tr>
<tr>
<td>Species</td>
<td>6</td>
<td>Varchar(22)</td>
</tr>
<tr>
<td>Pond Value</td>
<td>6</td>
<td>Numeric(8,3)</td>
</tr>
</tbody>
</table>

   (8 rows)

5. Set the EnableBetterFlexTypeGuessing parameter to 0 (OFF).

6. Call COMPUTE_FLEXTABLE.Keys with the trees flex table again.

7. Query the trees_keys table to compare the data_type_guess values with the previous results. Without the configuration parameter set, all of the non-string data types are VARCHARS of various lengths:

   ```sql
   => SELECT * FROM trees_keys;
   ```

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Quarter</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Region</td>
<td>6</td>
<td>varchar(66)</td>
</tr>
</tbody>
</table>

(8 rows)
8. To maintain accurate results for non-string data types, set the `EnableBetterFlexTypeGuessing` parameter back to 1 (ON).

For more information about setting the `EnableBetterFlexTypeGuessing` configuration parameter, see Setting Flex Table Configuration Parameters.

See Also

- `BUILD_FLEXTABLE_VIEW`
- `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`
- `MATERIALIZE_FLEXTABLE_COLUMNS`
- `RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW`

**COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**

Combines the functionality of `BUILD_FLEXTABLE_VIEW` and `COMPUTE_FLEXTABLE_KEYS` to compute virtual columns (keys) from the VMap data of a flex table and construct a view. Creating a view with this function ignores empty keys. If you do not need to perform both operations together, use one of the single-operation functions instead.

**Syntax**

`COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW('flex_table')`
Arguments

flex_table | The name of a flex table.

Examples

This example shows how to call the function for the darkdata flex table.

```sql
=> SELECT COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW('darkdata');
compute_flextable_keys_and_build_view

Please see public.darkdata_keys for updated keys
The view public.darkdata_view is ready for querying
(1 row)
```

See Also

- BUILD_FLEXTABLE_VIEW
- COMPUTE_FLEXTABLE_KEYS
- MATERIALIZE_FLEXTABLE_COLUMNS
- RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

**MATERIALIZE_FLEXTABLE_COLUMNS**

Materializes virtual columns listed as *key_names* in the *flextable_keys* table you compute using either COMPUTE_FLEXTABLE_KEYS or COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW.

Note: Each column that you materialize with this function counts against the data storage limit of your license. To check your Vertica license compliance, call the AUDIT() or AUDIT_FLEX() functions.

Syntax

MATERIALIZE_FLEXTABLE_COLUMNS('[[database.]schema.]flex-table' [, n-columns [, keys-table-name] ])}
## Arguments

| **database.**schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:  

```sql
myschema.thisDbObject
```

If you specify a database, it must be the current database. |
| --- | --- |
| **flex-table** | The name of the flex table with columns to materialize. Specifying only the flex table name attempts to materialize up to 50 columns of key names in the default flex_table_keys table. When you use this argument, the function:

- Skips any columns already materialized
- Ignores any empty keys

To materialize a specific number of columns, use the optional parameter n_columns, described next. |
| **n-columns** | The number of columns to materialize. The function attempts to materialize the number of columns from the flex_table_keys table, skipping any columns already materialized. 

Vertica tables support a total of 1600 columns, which is the largest value you can specify for n-columns. The function orders the materialized results by frequency, descending, key_name when materializing the first n columns. |
| **keys-table-name** | The name of a flex_keys_table from which to materialize columns. The function:

- Materializes the number of columns (value of n-columns) from keys-table-name
- Skips any columns already materialized
- Orders the materialized results by frequency, descending, key_name when materializing the first n columns. |
Examples

The following example shows how to call MATERIALIZE_FLEXTABLE_COLUMNS to materialize columns. First, load a sample file of tweets (tweets_10000.json) into the flex table twitter_r.

After loading data and computing keys for the sample flex table, call materialize_flextable_columns to materialize the first four columns:

```sql
=> COPY twitter_r FROM '/home/release/KData/tweets_10000.json' parser fjsonparser();
Rows Loaded

10000
(1 row)

=> SELECT compute_flextable_keys ('twitter_r');

compute_flextable_keys

Please see public.twitter_r_keys for updated keys

(1 row)

=> select materialize_flextable_columns('twitter_r', 4);

materialize_flextable_columns

The following columns were added to the table public.twitter_r:

| contributors | entities.hashtags | entities.urls |

For more details, run the following query:

```
selct * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and table_name = 'twitter_r';
```

(1 row)

The last message in the example recommends querying system table MATERIALIZE_FLEXTABLE_COLUMNS_RESULTS for the results of materializing the columns, as shown:

```sql
=> SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and table_name = 'twitter_r';
table_id | table_schema | table_name | creation_time | key_name | status | message

+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+------------------
| 4503596273733172 | public          | twitter_r      | 2013-11-20 17:00:27.945484-05 | contributors   | ADDED | Added successfully
| 4503596273733172 | public          | twitter_r      | 2013-11-20 17:00:27.945515-05 | entities.hashtags | ADDED | Added successfully
| 4503596273733172 | public          | twitter_r      | 2013-11-20 17:00:27.945519-05 | entities.urls | ADDED | Added successfully
| 4503596273733172 | public          | twitter_r      | 2013-11-20 17:00:27.945532-05 | created_at | EXISTS | Column of same name already
```

(4 rows)
See Also

- BUILD_FLEXTABLE_VIEW
- COMPUTE_FLEXTABLE_KEYS
- COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW
- RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

**RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW**

Restores the _keys table and the _view. The function also links the _keys table with its associated flex table, in cases where either table is dropped. The function also indicates whether it restored one or both objects.

**Syntax**

```
RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('flex_table')
```

**Arguments**

| flex_table          | The name of a flex table. |

**Examples**

This example shows how to invoke this function with an existing flex table, restoring both the _keys table and _view:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
```

```
RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata')
```

The keys table public.darkdata_keys was restored successfully.
The view public.darkdata_view was restored successfully.
(1 row)

This example illustrates that the function restored darkdata_view, but that darkdata_keys did not need restoring:

```
=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');

The keys table public.darkdata_keys already exists and is linked to darkdata.
The view public.darkdata_view was restored successfully.
(1 row)
```

After restoring the _keys table, there is no content. To populate the flex keys, call the
COMPUTE_FLEXTABLE.Keys meta function.

```
=> SELECT * FROM darkdata_keys;
key_name | frequency | data_type_guess
---------+-----------+------------------
(0 rows)
```

See Also

- `BUILD_FLEXTABLE_VIEW`
- `COMPUTE_FLEXTABLE_KEYS`
- `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`
- `MATERIALIZE_FLEXTABLE_COLUMNS`
Flex Extractor Functions Reference

The extractor scalar functions process polystructured data:
Each function accepts input data that is:

- Existing database content
- A table
- Returned from an expression
- Entered directly

These functions do not parse data from an external file source. All functions return a single VMap value. The extractor functions can return data with NULL-specified columns.

This section describes each extractor function.

MAPDELIMITEDEXTRACTOR

Extracts data with a delimiter character, and other optional arguments, returning a single VMap value. The USING PARAMETERS phrase specifies optional parameters for the function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>VARCHAR</td>
<td>Single delimiter character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong></td>
</tr>
<tr>
<td>header_names</td>
<td>VARCHAR</td>
<td>[Optional] Specifies header names for columns.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> ucoln</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where $n$ is the column offset number, starting with 0 for the first column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The function uses default values if you do not specify values for the header_names parameter.</td>
</tr>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Trims white space from header names and field values.</td>
</tr>
</tbody>
</table>
**Default value:** true

<table>
<thead>
<tr>
<th>treat_empty_val_as_null</th>
<th>BOOLEAN</th>
<th>[Optional] Specifies that empty fields become NULLs, rather than empty strings ('').</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
</tbody>
</table>

## Examples

These examples use a short set of delimited data:

```
Name|CITY|New city|State|zip
---|----|--------|-----|---
Tom|BOSTON|boston|MA|01
Eric|Burlington|BURLINGTON|MA|02
Jamie|cambridge|CAMBRIDGE|MA|08
```

To begin, save this data as delim.dat.

1. **Create a flex table, dflex:**

   ```sql
   => CREATE FLEX TABLE dflex();
   CREATE TABLE
   ```

2. **Use COPY to load the delim.dat file. Use the flex tables fdelimitedparser with the header='false' option:**

   ```sql
   => COPY dflex FROM '/home/release/kmm/flextables/delim.dat' parser fdelimitedparser (header='false');
   Rows Loaded
   ------------
   4
   (1 row)
   ```

3. **Create a columnar table, dtab, with an identity id column, a delim column, and a column to hold a VMap, named vmap:**

   ```sql
   => CREATE TABLE dtab (id IDENTITY(1,1), delim varchar(128), vmap long varbinary(512));
   CREATE TABLE
   ```

4. **Use COPY to load the delim.dat file into the dtab table. For the mapdelimitedextractor function, add a header row with USING PARAMETERS header_names= option to specify the header row for the sample data, along with delimiter '!':**

   ```sql
   ```
COPY dtab(delim, vmap AS MAPDELIMITEDEXTRACTOR (delim
USING PARAMETERS header_names='Name|CITY|New City|State|Zip')) FROM
'/home/dbadmin/data/delim.dat'
DELIMITER '!
;

Rows Loaded
---------------------
  4
(1 row)

5. Use maptostring for the flex table dflex to view the __raw__ column contents. Notice the default header names in use (ucol0 – ucol4), since you specified header='false' when you loaded the flex table:

SELECT MAPTOSTRING(__raw__) FROM dflex limit 10;

maptostring
-----------------------------------------------
{"ucol0": "Jamie",
 "ucol1": "cambridge",
 "ucol2": "CAMBRIDGE",
 "ucol3": "MA",
 "ucol4": "08"
}
{"ucol0": "Name",
 "ucol1": "CITY",
 "ucol2": "New city",
 "ucol3": "State",
 "ucol4": "zip"
}
{"ucol0": "Tom",
 "ucol1": "BOSTON",
 "ucol2": "boston",
 "ucol3": "MA",
 "ucol4": "01"
}
{"ucol0": "Eric",
 "ucol1": "Burlington",
 "ucol2": "BURLINGTON",
 "ucol3": "MA",
 "ucol4": "02"
}
(4 rows)

6. Use maptostring again, this time with the dtab table's vmap column. Compare the results of this output to those for the flex table. Note that maptostring returns the header_name parameter values you specified when you loaded the data:
7. Query the `delim` column to view the contents differently:

```sql
=> SELECT delim FROM dtab;

delim

Name|CITY|New city|State|zip
Tom|BOSTON|boston|MA|02121
Eric|Burlington|BURLINGTON|MA|02482
Jamie|cambridge|CAMBRIDGE|MA|02811
(4 rows)
```
See Also

- MAPJSONEXTRACTOR
- MAPREGEXEXTRACTOR

**MAPJSONEXTRACTOR**

Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements. The USING PARAMETERS phrase specifies optional parameters for the function. Empty input does not generate a Warning or Error.

Note: The function fails if the output size of the function is greater than 65000.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_maps</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.). Default value: true</td>
</tr>
<tr>
<td>flatten_arrays</td>
<td>BOOLEAN</td>
<td>[Optional] Converts lists to sub-maps with integer keys. Lists are not flattened by default. Default value: false</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues. Default value: false</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>BOOLEAN</td>
<td>[Optional] Rejects any row containing a key without a value (reject_on_empty_key=true). Default value: false</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>BOOLEAN</td>
<td>[Optional] Omits key from the load data that does not have a value (omit_empty_keys=true).</td>
</tr>
</tbody>
</table>
**Example**

These examples use the following sample JSON data:

```json
{
  "id": "5001", "type": "None"
}
{
  "id": "5002", "type": "Glazed"
}
{
  "id": "5005", "type": "Sugar"
}
{
  "id": "5007", "type": "Powdered Sugar"
}
{
  "id": "5004", "type": "Maple"
}
```

Save this example data as `bake_single.json`, and load that file.

1. Create a flex table, `flexjson`:

   ```sql
   => CREATE FLEX TABLE flexjson();
   CREATE TABLE
   ```

2. Use COPY to load the `bake_single.json` file with the `fjsonparser` parser:

   ```sql
   => COPY flexjson FROM '/home/dbadmin/data/bake_single.json' parser fjsonparser();
   Rows Loaded
   ------------
   5
   (1 row)
   ```

3. Create a columnar table, `coljson`, with an identity column (id), a json column, and a column to hold a VMap, called `vmap`:

   ```sql
   => CREATE TABLE coljson(id IDENTITY(1,1), json varchar(128), vmap long varbinary(10000));
   CREATE TABLE
   ```

4. Use COPY to load the `bake_single.json` file into the `coljson` table, using the `mapjsonextractor` function:
COPY coljson (json, vmap AS MapJSONExtractor(json)) FROM '/home/dbadmin/data/bake_single.json';
Rows Loaded
----------
   5
(1 row)

5. Use the maptoString function for the flex table flexjson to output the __raw__ column contents as strings:

```
=> SELECT MAPTOSTRING(__raw__) FROM flexjson limit 5;
```
```
<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>None</td>
</tr>
<tr>
<td>5002</td>
<td>Glazed</td>
</tr>
<tr>
<td>5005</td>
<td>Sugar</td>
</tr>
<tr>
<td>5007</td>
<td>Powdered Sugar</td>
</tr>
<tr>
<td>5004</td>
<td>Maple</td>
</tr>
</tbody>
</table>
```
(5 rows)

6. Use the maptoString function again, this time with the coljson table's vmap column and compare the results. The element order differs:

```
=> SELECT MAPTOSTRING(vmap) FROM coljson limit 5;
```
```
<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>None</td>
</tr>
<tr>
<td>5002</td>
<td>Glazed</td>
</tr>
<tr>
<td>5005</td>
<td>Sugar</td>
</tr>
<tr>
<td>5007</td>
<td>Powdered Sugar</td>
</tr>
<tr>
<td>5004</td>
<td>Maple</td>
</tr>
</tbody>
</table>
```
(5 rows)
See Also

- MAPDELIMITEDEXTRACTOR
- MAPREGEXEXTRACTOR

MAPREGEXEXTRACTOR

Extracts data from a regular expression and returns the results as a VMap. Use the USING PARAMETERS `pattern= phrase, followed by the regular expression.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pattern=</code></td>
<td>VARCHAR</td>
<td>The regular expression as a string. Default value: An empty string (&quot;&quot;&quot;).</td>
</tr>
<tr>
<td><code>use_jit</code></td>
<td>BOOLEAN</td>
<td>[Optional] Uses just-in-time compiling when parsing the regular expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: false.</td>
</tr>
<tr>
<td><code>record_terminator</code></td>
<td>VARCHAR</td>
<td>[Optional] The character used to separate input records. Default value: \n.</td>
</tr>
</tbody>
</table>
logline_column | VARCHAR
--- | ----
[Optional] The destination column containing the full string that the regular expression matched.
Default value: An empty string ("").

Examples

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, (<thread_id>(?:0x)?[0-9a-f]+).

```
^(?<time>\d{4}-\d{2}-\d{2}\d{2}:\d{2}:\d{3}.\d+)
(?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
(?<transaction_id>[0-9a-f]+)?(?:(?<component>\w+])\{|(?<level>\w+)\}
(?<text>.*)'
```

The following examples may include newline characters for display purposes.

1. Create a flex table, flogs:

   ```
   => CREATE FLEX TABLE flogs();
   CREATE TABLE
   ```

2. Use COPY to load a sample log file (vertica.log), using the flex table fregexparsr. Note that this example includes added line characters for displaying long text lines.

   ```
   => COPY flogs FROM '/home/dbadmin/tempdat/vertica.log' PARSER FREGEXPARSER(pattern='^
   ^(<time>\d{4}-\d{2}-\d{2}\d{2}:\d{2}:\d{3}.\d+) 
   (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+)
   (?<transaction_id>[0-9a-f]+)?(?:(?<component>\w+])\{|(?<level>\w+)\}
   (?<text>.*)');
   Rows Loaded
   81399
   (1 row)
   ```

3. Use MapToString to return the results from calling MapRegexExtractor with a regular expression. The output returns the results of the function in string format.

   ```
   => SELECT MAPTOSTRING(MapregexExtractor(E'2014-04-02 04:02:51.011
   TM Moveout:0x2aab9000f860-a000000002067 [Txn] <INFO>
   ```
Begin Txn: a0000000002067 'Moveout: Tuple Mover' using PARAMETERS
pattern='^(?<time>[\d\d]\d\d-\d\d-\d\d \d\d:\d\d:\d\d\.\d+)(?<thread_name>[A-Za-z ]+):(?:<thread_id>(?:0x)?[0-9a-f]+)-?(?:<transaction_id>[0-9a-f])?\ {[?:<component>[w+]]\<(?:<level>[w+])\> ?(?:<@?\<enode>[w+]?):
} )?\ (?<text>.+)) FROM flogs where __identity__=13
map
tostring
-----------------------------------------------------------------------------------------------------------------------------------
--
{
"component": "Txn",
"level": "INFO",
"text": "Begin Txn: a0000000002067 'Moveout: Tuple Mover'",
"thread_id": "0x2aab9000f860",
"thread_name": "TM Moveout",
"time": "2014-04-02 04:02:51.011",
"transaction_id": "a0000000002067"
}
(1 row)

See Also

- [MAPDELIMITEDEXTRACTOR](#)
- [MAPJSONEXTRACTOR](#)
Flex Map Functions Reference

The flex map functions let you extract and manipulate nested map data.

The first argument of all flex map functions (except emptymap() and mapaggregate()) takes a VMap. The VMap can originate from the __raw__ column in a flex table or be returned from a map or extraction function.

All map functions (except for emptymap() and mapaggregate()), accept either a LONG VARBINARY or a LONG VARCHAR map argument.

In the following example, the outer maplookup() function operates on the VMap data returned from the inner maplookup() function:

```sql
=> maplookup(maplookup(ret_map, 'batch'), 'scripts')
```

You can use flex map functions with:

- Flex tables
- Their associated `{flextable}_keys` table
- Automatically generated `{flextable}_view` views.

However, use of these functions does not apply to standard Vertica tables.

**EMPTYMAP**

Constructs a new VMap with one row but without keys or data. Use this transform function to populate a map without using a flex parser. Instead, you use either from SQL queries or from map data present elsewhere in the database.

**Syntax**

`EMPTYMAP()`

**Arguments**

None
Examples

Create an Empty Map

```sql
=> SELECT EMPTYMAP();
emptymap
--------------------------------------------------------------------------------
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

Create an Empty Map from an Existing Flex Table

If you create an empty map from an existing flex table, the new map has the same number of rows as the table from which it was created.

This example shows the result if you create an empty map from the `darkdata` table, which has 12 rows of JSON data:

```sql
=> SELECT EMPTYMAP() FROM darkdata;
emptymap
--------------------------------------------------------------------------------
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\001\000\000\004\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
(12 rows)
```

See Also

- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
MAPAGGREGATE

Returns a LONG VARBINARY VMap with keys and value pairs supplied from two VARCHAR input columns of an existing columnar table. Using this function requires specifying an over() clause for the source table.

Syntax

MAPAGGREGATE(source_column1, source_column2)

Arguments

<table>
<thead>
<tr>
<th>source_column1</th>
<th>Table column with values to use as the keys of the key/value pair of the returned VMap data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_column2</td>
<td>Table column with values to use as the values in the key/value pair of the returned VMap data.</td>
</tr>
</tbody>
</table>

Examples

This example creates a columnar table btest, with two VARCHAR columns, named keys and values, and adds three sets of values:

```sql
=> CREATE TABLE btest(keys varchar(10), values varchar(10));
CREATE TABLE
```
After populating the `btest` table, call `mapaggregate()` using the `OVER(PARTITION BEST)` clause. This call returns the raw_map data:

```sql
=> SELECT MAPAGGREGATE(keys, values) OVER(PARTITION BEST) FROM btest;
+-----------------------------------------------------------------------------+
| one|1  |
| two|2  |
| three|3  |
+-----------------------------------------------------------------------------+
(1 row)
```

The next example illustrates using `MAPTOSTRING()` with the returned raw_map from `mapaggregate()` to see the values:

```sql
=> SELECT MAPTOSTRING(raw_map) FROM (SELECT MAPAGGREGATE(keys, values) OVER(PARTITION BEST) FROM btest) bit;
+-----------------------------------------------------------------------------+
| {                                                                         |
|   "one": "1", "three": "3", "two": "2"  | |
+-----------------------------------------------------------------------------+
(1 row)
```

See Also

- `EMPTYMAP`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
MAPCONTAINSKEY

Determines whether a VMap contains a virtual column (key). This scalar function returns true (t), if the virtual column exists, or false (f) if it does not. Determining that a key exists before calling maplookup() lets you distinguish between NULL returns. The maplookup() function uses for both a non-existent key and an existing key with a NULL value.

Syntax

MAPCONTAINSKEY(VMap_data, 'virtual_column_name')

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- The <em><strong>raw</strong></em> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>- Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>- Other database content</td>
</tr>
<tr>
<td>virtual_column_name</td>
<td>The name of the key to check.</td>
</tr>
</tbody>
</table>
Examples

This example shows how to use the mapcontainskey() functions with maplookup(). View the results returned from both functions. Check whether the empty fields that maplookup() returns indicate a NULL value for the row (t) or no value (f):

You can use mapcontainskey() to determine that a key exists before calling maplookup(). The maplookup() function uses both NULL returns and existing keys with NULL values to indicate a non-existent key.

```sql
=> SELECT MAPLOOKUP(_raw__, 'user.location'), MAPCONTAINSKEY(_raw__, 'user.location')
FROM darkdata ORDER BY 1;
maplookup | mapcontainskey
-----------|------------------------
     t    |           t           
     t    |           t           
     t    |           t           
   Chile  |           t           
   Narnia |           t           
  Uptown..|           t           
  chicago |           f           
          |           f           
          |           f           
          |           f           
(12 rows)
```

See Also

- [EMPTYMAP](#)
- [MAPAGGREGATE](#)
- [MAPCONTAINSVALUE](#)
- [MAPITEMS](#)
- [MAPKEYS](#)
- [MAPKEYSINFO](#)
- [MAPLOOKUP](#)
- [MAPSIZE](#)
MAPCONTAINSVALUE

Determines whether a VMap contains a specific value. Use this scalar function to return true (t), if the value exists, or false (f), if it does not.

Syntax

MAPCONTAINSVALUE(VMap_data, 'virtual_column_value')

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMap_data</td>
<td>Any VMap data. The VMap can exist as:</td>
</tr>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
<tr>
<td>virtual_column_value</td>
<td>The value whose existence you want to confirm.</td>
</tr>
</tbody>
</table>

Examples

This example shows how to use mapcontainsvalue() to determine whether or not a virtual column contains a particular value. Create a flex table (ftest), and populate it with some virtual columns and values. Name both virtual columns one:

```sql
=> CREATE FLEX TABLE ftest();
CREATE TABLE
=> copy ftest from stdin parser fjsonparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
```
Call `mapcontainsvalue()` on the `ftest` map data. The query returns false (f) for the first virtual column, and true (t) for the second, which contains the value one:

```sql
=> SELECT MAPCONTAINSVALUE(\_raw\_, 'one') FROM ftest;
mapcontainsvalue
------------------
f t
(2 rows)
```

### See Also

- `EMPTYMAP`
- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPITEMS`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
- `MAPSIZE`
- `MAPTOSTRING`
- `MAPVALUES`
- `MAPVERSION`

### MAPITEMS

Returns information about items in a VMap. Use this transform function with one or more optional arguments to access polystructured values within the VMap data. This function requires an `OVER()` clause.
Syntax

MAPITEMS(VMap_data [, passthrough_arg [, ...]] )

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMap_data</td>
<td>Any VMap data. The VMap can exist as:</td>
</tr>
<tr>
<td></td>
<td>• The <em><strong>raw</strong></em> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
<tr>
<td>passthrough_arg</td>
<td>[Optional] One or more arguments indicating keys within the map data in VMap_data</td>
</tr>
</tbody>
</table>

Examples

The following examples illustrate using mapItems() with the over(PARTITION BEST) clause.

This example determines the number of virtual columns in the map data using a flex table, labeled darkmountain. Query using the count() function to return the number of virtual columns in the map data:

```sql
=> SELECT COUNT(keys) FROM (SELECT MAPITEMS(darkmountain.__raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;

count
------
19

(1 row)
```

The next example determines what items exist in the map data:

```sql
=> SELECT * FROM (SELECT MAPITEMS(darkmountain.__raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;

keys | values
------|--------
hike_safety | 50.6
name | Mt Washington
type | mountain
height | 17000
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Height</th>
<th>Hike Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denali</td>
<td>Mountain</td>
<td>29029</td>
<td>12.2</td>
</tr>
<tr>
<td>Everest</td>
<td>Mountain</td>
<td>14000</td>
<td>34.1</td>
</tr>
<tr>
<td>Kilimanjaro</td>
<td>Mountain</td>
<td>29029</td>
<td>22.8</td>
</tr>
<tr>
<td>Mt St Helens</td>
<td>Volcano</td>
<td></td>
<td>15.4</td>
</tr>
</tbody>
</table>

**Directly Query a Key Value in a VMap**

Review the following JSON input file, `simple.json`. In particular, notice the array called `three_Array`, and its four values:

```json
{
  "one": "one",
  "two": 2,
  "three_Array": [
    "three_One",
    "three_Two",
    3,
    "three_Four"
  ],
  "four": 4,
  "five_Map": {
    "five_One": 51,
    "five_Two": "Fifty-two",
    "five_Three": "fifty three",
    "five_Four": 54,
    "five_Five": "5 x 5"
  },
  "six": 6
}
```

1. **Create a flex table, mapper:**

```sql
=> CREATE FLEX TABLE mapper();
CREATE TABLE
```

1. **Load simple.json into the flex table mapper:**

```sql
=> COPY mapper FROM '/home/dbadmin/data/simple.json' parser fjsonparser (flatten_arrays=false, flatten_maps=false);
Rows Loaded
----------
```
2. Call mapkeys on the flex table's __raw__ column to see the flex table's keys, but not the key submaps. The return values indicate three_Array as one of the virtual columns:

```sql
=> SELECT MAPKEYS(__raw__) OVER() FROM mapper;
keys
----------
five_Map
four
one
six
three_Array
two
(6 rows)
```

3. Call mapitems on flex table mapper with three_Array as a pass-through argument to the function. The call returns these array values:

```sql
=> SELECT __identity__, mapitems(three_Array) OVER(PARTITION BY __identity__) FROM mapper;
__identity__ | keys | values
----------------
1 | 0 | three_One
1 | 1 | three_Two
1 | 2 | 3
1 | 3 | three_Four
(4 rows)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
MAPKEYS

Returns the virtual columns (and values) present in any VMap data. This transform function requires an over(PARTITION BEST) clause.

Syntax

MAPKEYS(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

Examples

Determine Number of Virtual Columns in Map Data

This example shows how to create a query, using an over(PARTITION BEST) clause with a flex table, darkdata to find the number of virtual column in the map data. The table is populated with JSON tweet data.

=> SELECT COUNT(keys) FROM (SELECT MAPKEYS(darkdata.__raw__) OVER(PARTITION BEST) FROM darkdata) AS a;
Query Ordered List of All Virtual Columns in the Map

This example shows a snippet of the return data when you query an ordered list of all virtual columns in the map data:

```sql
=> SELECT * FROM (SELECT MAPKEYS(darkdata._raw__) OVER(PARTITION BEST) FROM darkdata) AS a;
```

<table>
<thead>
<tr>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
</tr>
<tr>
<td>coordinates</td>
</tr>
<tr>
<td>created_at</td>
</tr>
<tr>
<td>delete.status.id</td>
</tr>
<tr>
<td>delete.status.id_str</td>
</tr>
<tr>
<td>delete.status.user_id</td>
</tr>
<tr>
<td>delete.status.user_id_str</td>
</tr>
<tr>
<td>entities.hashtags</td>
</tr>
<tr>
<td>entities.media</td>
</tr>
<tr>
<td>entities.urls</td>
</tr>
<tr>
<td>entities.user_mentions</td>
</tr>
<tr>
<td>favorited</td>
</tr>
<tr>
<td>geo</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>user.statuses_count</td>
</tr>
<tr>
<td>user.time_zone</td>
</tr>
<tr>
<td>user.url</td>
</tr>
<tr>
<td>user.utc_offset</td>
</tr>
<tr>
<td>user.verified</td>
</tr>
</tbody>
</table>

(125 rows)

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYSINFO
MAPKEYSINFO

Returns virtual column information from a given map. This transform function requires an `over(PARTITION BEST) clause.

Syntax

MAPKEYSINFO(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <code>__raw__</code> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

Returns

This function is a superset of the MAPKEYS() function. It returns the following information about each virtual column:
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>The virtual column names in the raw data.</td>
</tr>
<tr>
<td>length</td>
<td>The data length of the key name, which can differ from the actual string length.</td>
</tr>
<tr>
<td>type_oid</td>
<td>The OID type into which the value should be converted. Currently, the type is always 116 for a LONG VARCHAR, or 199 for a nested map that is stored as a LONG VARBINARY.</td>
</tr>
<tr>
<td>row_num</td>
<td>The number of rows in which the key was found.</td>
</tr>
<tr>
<td>field_num</td>
<td>The field number in which the key exists.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows a snippet of the return data you receive if you query an ordered list of all virtual columns in the map data:

```sql
=> SELECT * FROM (SELECT MAPKEYSINFO(darkdata.__raw__) OVER(PARTITION BEST) FROM darkdata) AS a;
```

<table>
<thead>
<tr>
<th>keys</th>
<th>length</th>
<th>type_oid</th>
<th>row_num</th>
<th>field_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>coordinates</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>created_at</td>
<td>30</td>
<td>116</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>entities.hashtags</td>
<td>93</td>
<td>199</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>entities.media</td>
<td>772</td>
<td>199</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>entities.urls</td>
<td>16</td>
<td>199</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>entities.user.mentions</td>
<td>16</td>
<td>199</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>favorited</td>
<td>1</td>
<td>116</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>geo</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>id</td>
<td>18</td>
<td>116</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>id_str</td>
<td>18</td>
<td>116</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>delete.status.id</td>
<td>18</td>
<td>116</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>delete.status.id_str</td>
<td>18</td>
<td>116</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>delete.status.user_id</td>
<td>9</td>
<td>116</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>delete.status.user_id_str</td>
<td>9</td>
<td>116</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>delete.status.id</td>
<td>18</td>
<td>116</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>delete.status.id_str</td>
<td>18</td>
<td>116</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>delete.status.user_id</td>
<td>9</td>
<td>116</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>delete.status.user_id_str</td>
<td>9</td>
<td>116</td>
<td>12</td>
<td>3</td>
</tr>
</tbody>
</table>

(550 rows)
See Also

- `EMPTYMAP`
- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
- `MAPLOOKUP`
- `MAPSIZE`
- `MAPTOSTRING`
- `MAPVALUES`
- `MAPVERSION`

## MAPLOOKUP

Returns single-key values from VMAP data. This scalar function returns a `LONG VARCHAR`, with values, or `NULL` if the virtual column does not have a value.

Using `maplookup` is case insensitive to virtual column names. To avoid loading same-name values, set the `fjsonparser` parser `reject_on_duplicate` parameter to `true` when data loading.

You can control the behavior for non-scalar values in a VMAP (like arrays), when loading data with the `fjsonparser` or `favroparser` parsers and its `flatten-arrays` argument. See [Loading JSON Data](#) and the `FJSONPARSER` reference.

For information about using `maplookup()` to access nested JSON data, see [Querying Nested Data](#).
Syntax

MAPLOOKUP(VMap_data, 'virtual_column_name' [USING PARAMETERS [case_sensitive={false | true}] [, buffer_size=n] ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMap_data</td>
<td>Any VMap data. The VMap can exist as:</td>
</tr>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
<tr>
<td>virtual_column_name</td>
<td>The name of the virtual column whose values this function returns.</td>
</tr>
<tr>
<td>buffer_size</td>
<td>[Optional parameter] Specifies the maximum length (in bytes) of each value returned for virtual_column_name. To return all values for virtual_column_name, specify a buffer_size equal to or greater than (=&gt;) the number of bytes for any returned value. Any returned values greater in length than buffer_size are rejected.</td>
</tr>
<tr>
<td></td>
<td>Default value: 0 (No limit on buffer_size)</td>
</tr>
<tr>
<td>case_sensitive</td>
<td>[Optional parameter]</td>
</tr>
<tr>
<td></td>
<td>Specifies whether to return values for virtual_column_name if keys with different cases exist.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>(... USING PARAMETERS case_sensitive=true)</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
</tbody>
</table>

Examples

This example returns the values of one virtual column, user.location:
Using maplookup buffer_size

Use the buffer_size= parameter to indicate the maximum length of any value that maplookup returns for the virtual column you specify. If none of the returned key values can be greater than n bytes, use this parameter to allocate n bytes as the buffer_size.

For the next example, save this JSON data to a file, simple_name.json:

```json
{
  "name": "sierra",
  "age": "63",
  "eyes": "brown",
  "weapon": "doggie"
}
{
  "name": "janis",
  "age": "10",
  "eyes": "blue",
  "weapon": "humor"
}
{
  "name": "ben",
  "age": "43",
  "eyes": "blue",
  "weapon": "sword"
}
{
  "name": "jen",
  "age": "38",
  "eyes": "green",
  "weapon": "shopping"
}
```

1. Create a flex table, logs.

2. Load the simple_name.json data into logs, using the fjsonparser. Specify the flatten_arrays option as True:

```sql
=> SELECT MAPLOOKUP(_raw_, 'user.location') FROM darkdata ORDER BY 1;
maplookup
---------
Chile
Nesnia
Uptown

chicago
(12 rows)
```

```sql
COPY logs FROM '/home/dbadmin/data/simple_name.json' PARSER fjsonparser(flatten_arrays=True);
```
3. Use `maplookup` with `buffer_size=0` for the `logs` table `name` key. This query returns all of the values:

```sql
=> SELECT MAPLOOKUP(_raw__, 'name' USING PARAMETERS buffer_size=0) FROM logs;
MapLookup
---------
sierra
ben
janis
jen
(4 rows)
```

4. Next, call `maplookup()` three times, specifying the `buffer_size` parameter as 3, 5, and 6, respectively. Now, `maplookup()` returns values with a byte length less than or equal to (`<=`) `buffer_size`:

```sql
=> SELECT MAPLOOKUP(_raw__, 'name' USING PARAMETERS buffer_size=3) FROM logs;
MapLookup
---------

  ben
  jen
(4 rows)

=> SELECT MAPLOOKUP(_raw__, 'name' USING PARAMETERS buffer_size=5) FROM logs;
MapLookup
---------

  janis
  jen
  ben
(4 rows)

=> SELECT MAPLOOKUP(_raw__, 'name' USING PARAMETERS buffer_size=6) FROM logs;
MapLookup
---------

  sierra
  janis
  jen
  ben
(4 rows)
```

**Disambiguate Empty Output Rows**

This example shows how to interpret empty rows. Using `maplookup` without first checking whether a key exists can be ambiguous. When you review the following output, 12 empty rows, you cannot determine whether a `user.location` key has:

- A non-NULL value
- A NULL value
- No value
To disambiguate empty output rows, use the mapcontainskey() function in conjunction with maplookup(). When maplookup returns an empty field, the corresponding value from mapcontainskey indicates t for a NULL or other value, or f for no value.

The following example output using both functions lists rows with NULL or a name value as t, and rows with no value as f:

```
=> SELECT MAPLOOKUP(__raw__, 'user.location'), MAPCONTAINSKEY(__raw__, 'user.location')
FROM darkdata ORDER BY 1;
maplookup | mapcontainskey
-----------|--
   t       | t
   t       | t
   t       | t
Chile     | t
Nesnia    | t
Uptown    | t
chicago   | t
   f       | No value
   f       | No value
   f       | No value
   f       | No value
(12 rows)
```

Check for Case-Sensitive Virtual Columns

You can use maplookup() with the case_sensitive parameter to return results when key names with different cases exist.

1. Save the following sample content as a JSON file. This example saves the file as repeated_key_name.json:

   ```json
   {
     "test": "lower1"
   }
   ```
2. Create a flex table, dupe, and load the JSON file:

```sql
CREATE FLEX TABLE dupe();
CREATE TABLE dbt=
COPY dupe FROM '/home/release/KData/repeated_key_name.json' parser fjsonparser();
Rows Loaded
--------------
    8
(1 row)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
MAPSIZE

Returns the number of virtual columns present in any VMap data. Use this scalar function to determine the size of keys.

Syntax

MAPSIZE(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>
Examples

This example shows the returned sizes from the number of keys in the flex table darkmountain:

```sql
=> SELECT MAPSIZE(__raw__) FROM darkmountain;
mapsize
-------
 3
 4
 4
 4
 4
(5 rows)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSTRING
- MAPVALUES
- MAPVERSION
**MAPTOSTRING**

Recursively builds a string representation VMap data, including nested JSON maps. Use this transform function to display the VMap contents in a readable LONG VARCHAR format. Use maptostring to see how map data is nested before querying virtual columns with mapvalues().

**Syntax**

```
MAPTOSTRING(VMap_data [using parameters canonical_json={true | false}])
```

**Arguments**

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <code>__raw__</code> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as <code>maplookup()</code></td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>canonical_json</th>
<th><code>=bool</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Optional parameter]</td>
<td>Produces canonical JSON output by default, using the first instance of any duplicate keys in the map data.</td>
</tr>
</tbody>
</table>

Use this parameter as other UDF parameters, preceded by using parameters, as shown in the examples. Setting this argument to false maintains the previous behavior of `maptostring()` and returns same-name keys and their values.

**Default value:** `canonical-json=true`
Examples

The following example shows how to create a sample flex table, darkdata and load JSON data from STDIN. By calling maptostring() twice with both values for the canonical_json parameter, you can see the different results on the flex table __raw__ column data.

1. Create sample table:

```sql
=> CREATE FLEX TABLE darkdata();
```

2. Load sample JSON data from STDIN:

```sql
=> COPY darkdata FROM stdin parser fjsonparser;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> { "aaa": 1, "aaa": 2, "AAA": 3, "bbb": "aaa\"bbb"
>> \.
```

3. Call maptostring() with its default behavior using canonical JSON output, and then review the flex table contents. The function returns the first duplicate key and its value ("aaa": "1") but omits remaining duplicate keys ("aaa": "2"):

```sql
=> SELECT MAPTOSTRING(__raw__) FROM darkdata;

maptostring
------------------------------------------
{  
   "AAA" : "3",
   "aaa" : "1",
   "bbb" : "aaa\"bbb"
}
(1 row)
```

4. Next, call maptostring() with using parameters canonical_json=false). This time, the function returns the first duplicate keys and their values:

```sql
=> SELECT MAPTOSTRING(__raw__ using parameters canonical_json=false) FROM darkdata;

maptostring
------------------------------------------
{  
   "aaa": "1",
   "aaa": "2",
   "AAA": "3",
   "bbb": "aaa\"bbb"
}
```
See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPVALUES
- MAPVERSION

**MAPVALUES**

Returns a string representation of the top-level values from a VMap. This transform function requires an over ( ) clause.

**Syntax**

`MAPVALUES(VMap_data)`
Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>The VMap from which values should be returned. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

Examples

The following example shows how to query a darkmountain flex table, using an over() clause (in this case, the over(PARTITION BEST) clause) with mapvalues().

```sql
=> SELECT * FROM (SELECT MAPVALUES(darkmountain.__raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;
values
------------------
 29029  34.1
Everest mountain 29029  15.4
Mt St Helens volcano 17000  12.2
Denali mountain 14000  22.8
Kilimanjaro mountain 50.6
Mt Washington mountain (19 rows)
```
See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPTOSTRING
- MAPVERSION

**MAPVERSION**

Returns the version or invalidity of any map data. This scalar function returns the map version (such as 1) or -1, if the map data is invalid.

**Syntax**

MAPVERSION(VMap_data)

**Arguments**

| VMap_data | The VMap data either from a __raw__ column in a flex table or from the data returned from a map function such as maplookup(). |
Examples

The following example shows how to use mapversion() with the darkmountainflex table, returning mapversion 1 for the flex table map data:

```sql
=> SELECT MAPVERSION(__raw__) FROM darkmountain;

mapversion
-----------
  1
  1
  1
  1
  1

(5 rows)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPTOSTRING
- MAPVALUES
Flex Parsers Reference

Vertica supports several parsers to load different types of data into flex tables.

Unlike with columnar tables, you must specify which parser to use when loading flex tables. You can use each flex parser to load the parser's associated type of data into columnar tables.

All parsers store the data as a single Vmap in the LONG VARCHAR column. If a data row is too large to fit in the column, it is rejected. Vertica supports null values for loading data with NULL-specified columns.

For information about how you can use each type of flex parser, see Using Flex Table Parsers

FAVROPARSER

Parses data from an Avro file. The input file must use binary serialization encoding. Use this parser to load data into columnar, flex, and hybrid tables.

Note: The parser favroparser does not support Avro files with separate schema files. The Avro file must have its related schema in the file you are loading.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_arrays</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens all Avro arrays. Key names are concatenated with nested levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false (Arrays are not flattened.)</td>
</tr>
<tr>
<td>flatten_maps</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens all Avro maps. Key names are concatenated with nested levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
<tr>
<td>flatten_records</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens all Avro records. Key names are concatenated with nested levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
</tbody>
</table>
| reject_on_materialized_type_error | BOOLEAN | [Optional] Indicates whether to reject any row value for a materialized column that }
the parser cannot coerce into a compatible data type. See Using Flex Table Parsers.

Default value: false

Examples

This example shows how to create and load a flex table with Avro data using favroparser. After loading the data, you can query virtual columns.

1. Create a flex table for Avro data, avro_basic:

```sql
=> CREATE FLEX TABLE avro_basic();
CREATE TABLE
```

2. Use the favroparser to load the data from an Avro file (weather.avro).

```sql
=> COPY avro_basic FROM '/home/dbadmin/data/flexcsv/weather.avro' PARSER favroparser();
Rows Loaded
---------------
5
(1 row)
```

3. Query virtual columns from the avro flex table:

```sql
=> SELECT station, temp, time FROM avro_basic;
station | temp | time
--------------
mohali | 0 | -619524000000
lucknow | 22 | -619506000000
norwich | -11 | -619484400000
ams | 111 | -655531200000
baddi | 78 | -655509600000
(5 rows)
```

For more information, see Loading Avro Data.

See Also

- FCEFPARSER
- FCSVPARSER
- FDELIMITEDPARSER
FCEFPARSER

Parses OpenText ArcSight Common Event Format (CEF) log files. The fcefparsr loads values directly into any table column with a column name that matches a source data key. The parser stores the data loaded into a flex table in a single VMap.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>CHAR</td>
<td>[Optional] Specifies a single-character delimiter. Default value: '\n'</td>
</tr>
<tr>
<td>record_terminator</td>
<td>CHAR</td>
<td>[Optional] Specifies a single-character record terminator. Default value: newline</td>
</tr>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Trims white space from header names and key values. Default value: true</td>
</tr>
<tr>
<td>reject_on_unescaped_delimiter</td>
<td>BOOLEAN</td>
<td>[Optional] Determines whether to reject rows containing unescaped delimiters. The CEF standard does not permit them. Default value: false</td>
</tr>
</tbody>
</table>

Examples

The following example illustrates creating a sample flex table for CEF data, with two real columns, eventId and priority.
1. **Create a flex table cefdata:**

```sql
=> create flex table cefdata();
CREATE TABLE
```

2. **Load some basic CEF data, using the flex parser fcefparser:**

```sql
=> copy cefdata from stdin parser fcefparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> CEF:0|ArcSight|ArcSight|2.4.1|machine:20|New alert|High
>> .
```

3. **Use the maptostring() function to view the contents of your cefdata flex table:**

```sql
=> select maptostring(_raw__) from cefdata;
maptostring
+-----------------------------------------------
 | "deviceproduct" : "ArcSight",
 | "devicevendor" : "ArcSight",
 | "deviceversion" : "2.4.1",
 | "name" : "New alert",
 | "severity" : "High",
 | "signatureid" : "machine:20",
 | "version" : "0"
} (1 row)
```

4. **Select some virtual columns from the cefdata flex table:**

```sql
=> select deviceproduct, severity, deviceversion from cefdata;
deviceproduct | severity | deviceversion
--------------|----------|---------------
ArcSight      | High     | 2.4.1
(1 row)
```

For more information, see [Loading Common Event Format (CEF) Data](#)

**See Also**

- FAVROPARSER
- FCSVPARSER
**FCSVPARSER**

Parses CSV format (comma-separated values) data. Use this parser to load CSV data into columnar, flex, and hybrid tables. All data must be encoded in Unicode UTF-8 format. The parser `fcsvparser` supports the RFC 4180 de facto standard for CSV data, and other options, to accommodate variations in CSV file format definitions. Invalid records will be rejected.

The `fcsvparser` does not support multibyte data. For more information about data formats, see Checking Data Format Before or After Loading.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type</code></td>
<td>CHAR</td>
<td>[Optional] Specifies the default parameter values for the parser.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You do not have to use the type parameter when loading data that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conforms to the RFC 4180 standard (such as MS Excel files). See Loading CSV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data for the RFC4180 default parameters, and other options you can specify</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for traditional CSV files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> RFC4180</td>
</tr>
<tr>
<td><code>delimiter</code></td>
<td>CHAR</td>
<td>[Optional] Indicates the single-character value used to separate fields in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the CSV data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> , (for RFC4180 and traditional)</td>
</tr>
<tr>
<td><code>escape</code></td>
<td>CHAR</td>
<td>[Optional] Specifies a single-character</td>
</tr>
</tbody>
</table>
value. Use an escape character to interpret the next character in the data literally.

**Default value:** " (for rfc4180)

**Default value:** \ (for traditional)

### enclosed_by

**CHAR**  
[Optional] Specifies a single-character value. Use and enclosed_by value to include a value that is identical to the delimiter, but should be interpreted literally. For example, if the data delimiter is a comma (,), and you want to use a comma within the data ("my name is jane, and his is jim").

**Default value:** " (for rfc4180 and traditional)

### record_terminator

**CHAR**  
[Optional] Indicates the single-character value used to specify the end of a record.

**Default value:** \n or \r\n (for rfc4180 and traditional)

### header

**BOOLEAN**  
[Optional] Specifies whether to use the first row of data as a header column. When header=true (default), and no header exists, fcsvparser uses a default column heading. The default header consists of ucol$n$, where $n$ is the column offset number, starting with 0 for the first column. You can specify custom column heading names using the header_names parameter, described next.

If you specify header=false, the fcsvparser parses the first row of input as data, rather than as column headers.

**Default value:** true

### header_names

**CHAR**  
[Optional] Specifies a list of column header names, delimited by the character defined by the parser's delimiter parameter. Use
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates whether to trim white space from header names and key values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates how the parser handles header keys without values. If omit_empty_keys=true, keys with an empty value in the header row are not loaded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to reject any row containing a key without a value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_materialized_type_error</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates whether to reject any materialized column value that the parser cannot coerce into a compatible data type. See Loading CSV Data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
</tbody>
</table>

### Examples

This example shows how you can use fcsvparser to load a flex table, build a view, and then query that view.
1. Create a flex table for CSV data:

```sql
=> CREATE FLEX TABLE rfc();
```

2. Use `fcsvparser` to load the data from STDIN. Specify that no header exists, and enter some data as shown:

```sql
=> COPY rfc FROM stdin PARSER fcsvparser(header='false');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
> 10,10,20
> 10,"10",30
> 10,20"5",90
> .
```

3. Run the `compute_flextable_keys_and_build_view` function, and query the `rfc_view`. Notice that the default enclosed_by character permits an escape character (" ) within a field ("20""5"). Thus, the resulting value was parsed correctly. Since no header existed in the input data, the function added `ucoln` for each column:

```sql
=> SELECT compute_flextable_keys_and_build_view('rfc');
```

Please see public.rfc_keys for updated keys
The view public.rfc_view is ready for querying
(1 row)

```sql
=> SELECT * FROM rfc_view;
```

```sql
<table>
<thead>
<tr>
<th>ucol0</th>
<th>ucol1</th>
<th>ucol2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>20&quot;5&quot;</td>
<td>90</td>
</tr>
</tbody>
</table>
```

(3 rows)

For more information and examples of using other parameters of this parser, see Loading CSV Data.
See Also

- FAVROPARSER
- FDELIMITEDPAIRPARSER
- FDELIMITEDPARSER
- FDELIMITEDPAIRPARSER
- FJSONPARSER
- FJONSPARSER
- FREGEXPARSER

**FDELIMITEDPAIRPARSER**

Parses delimited data files. This parser provides a subset of the functionality in the parser fdelimitedparser. Use the fdelimitedpairparser when the data you are loading specifies pairs of column names with data in each row.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>CHAR</td>
<td>[Optional] Specifies a single-character delimiter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> <code>' '</code></td>
</tr>
<tr>
<td>record_terminator</td>
<td>CHAR</td>
<td>[Optional] Specifies a single-character record terminator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> newline</td>
</tr>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Trims white space from header names and key values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
</tbody>
</table>
Examples

The following example illustrates creating a sample flex table for simple delimited data, with two real columns, `eventId` and `priority`.

1. Create a table:

```sql
=> create flex table CEFData(eventId int default(eventId::int), priority int default (priority::int));
CREATE TABLE
```

2. Load a sample delimited OpenText ArcSight log file into the CEFData table, using the `fcefparser`:

```sql
=> copy CEFData from '/home/release/kmm/flextables/sampleArcSight.txt' parser fdelimitedpairparser();
Rows Loaded | 200
```

3. After loading the sample data file, use `maptostring()` to display the virtual columns in the `__raw__` column of CEFData:

```sql
=> select maptostring(__raw__) from CEFData limit 1;
```

```
maptostring
-----------------------------------------------
"agentassetid": "4-WWhuD08ABCCQ0VaeX21vg==",  
"agentzone": "3083",  
"agt": "265723237",  
"ghost": "svsvm0176",  
"aid": "3tGoHuD08ABCCM0VaeX21vg==",  
"art": "1099267576901",  
"assetcriticality": "0",  
"at": "snort_db",  
"atz": "America/Los_Angeles",  
"av": "5.3.0.19524.0",  
"cat": "attempted-recon",  
"categorybehavior": "/Communicate/Query",  
"categorydevicegroup": "/IDS/Network",  
"categoryobject": "/Host",  
"categoryoutcome": "/Attempt",  
"categoriesignificance": "/Recon",  
"categorytechnique": "/Scan",  
"categorytupledescription": "An IDS observed a scan of a host.",  
"cnt": "1",  
"cs2": "3",  
"destinationgeocountrycode": "US",  
"destinationgeolocationinfo": "Richardson",
```
4. **Select the eventID and priority real columns, along with two virtual columns, atz and destinationgeoregioncode:**

```sql
"destinationgeopostalcode" : "75082",
"destinationgeoregioncode" : "TX",
"destinationzone" : "3133",
"device product" : "Snort",
"device vendor" : "Snort",
"device version" : "1.8",
"deviceseverity" : "2",
"dhost" : "198.198.121.200",
"diat" : "329913940429",
"dlong" : "-966644973754",
"dst" : "3334896072",
"dtz" : "America/Los_Angeles",
"dvchost" : "unknown:eth1",
"end" : "1364676323451",
"eventid" : "1219383333",
"fdevice product" : "Snort",
"fdevice vendor" : "Snort",
"fdevice version" : "1.8",
"fdtz" : "America/Los_Angeles",
"fdvchost" : "unknown:eth1",
"lblstring2label" : "sig_rev",
"locality" : "0",
"modelconfidence" : "0",
"mrt" : "1364675789222",
"name" : "ICMP PING NMAP",
"oagentassetid" : "4-WwhuD0BABCCQGVAeX21vG==",
"oagentzone" : "3083",
"oagt" : "265723237",
"oahost" : "svsvm0176",
"oaid" : "3tGoHuD0BABCMDVAeX21vG==",
"oat" : "snort_db",
"oatz" : "America/Los_Angeles",
"oav" : "5.3.0.19524.0",
"originator" : "0",
"priority" : "8",
"proto" : "ICMP",
"relevance" : "10",
"rt" : "189267573000",
"severity" : "8",
"shost" : "198.198.104.10",
"signature id" : "[1:469]",
"slat" : "329913940429",
"slong" : "-966644973754",
"sourcegeocountrycode" : "US",
"sourcegeolocationinfo" : "Richardson",
"sourcegeopostalcode" : "75082",
"sourcegeoregioncode" : "TX",
"sourcezone" : "3133",
"src" : "3334891530",
"start" : "1364676323451",
"type" : "0"
}
```

(1 row)
```sql
SELECT eventID, priority, atz, destinationgeoregioncode FROM CEFData LIMIT 10;
```

<table>
<thead>
<tr>
<th>eventID</th>
<th>priority</th>
<th>atz</th>
<th>destinationgeoregioncode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1218325417</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td></td>
</tr>
<tr>
<td>1219383333</td>
<td>8</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1219533691</td>
<td>9</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220034458</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220034578</td>
<td>9</td>
<td>America/Los_Angeles</td>
<td></td>
</tr>
<tr>
<td>1220067119</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220106960</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220142122</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220312009</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>TX</td>
</tr>
<tr>
<td>1220321355</td>
<td>5</td>
<td>America/Los_Angeles</td>
<td>CA</td>
</tr>
</tbody>
</table>

(10 rows)

See Also

- FAVROPARSER
- FCEFPARSER
- FCSVPARSER
- FDELIMITEDPARSER
- FJSONPARSER
- FREGEXPARSER

**FDELIMITEDPARSER**

Parses data using a delimiter character to separate values. The `fdelimitedparser` loads delimited data, storing it in a single-value VMap. You can use this parser to load data into columnar and flex tables.

**Note:** By default, `fdelimitedparser` treats empty fields as NULL, rather than as an empty string (`''`). This behavior makes casting easier. Casting a NULL to an integer (`NULL::int`) is valid, while casting an empty string to an integer (`''::int`) is not. If required, use the `treat_empty_val_as_null` parameter to change the default behavior of `fdelimitedparser`. 
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>CHAR</td>
<td>[Optional] Indicates a single-character delimiter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong></td>
</tr>
<tr>
<td>record_terminator</td>
<td>CHAR</td>
<td>[Optional] Indicates a single-character record terminator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> \n</td>
</tr>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Determines whether to trim white space from header names and key values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
<tr>
<td>header</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies that a header column exists. The parser uses col### for the column names if you use this parameter but no header exists.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates how the parser handles header keys without values. If omit_empty_keys=true, keys with an empty value in the header row are not loaded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to reject any row containing a key without a value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> false</td>
</tr>
<tr>
<td>reject_on_materialized_type_error</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates whether to reject any</td>
</tr>
</tbody>
</table>
row value for a materialized column that the parser cannot coerce into a compatible data type. See Using Flex Table Parsers.

**Default value:** false

| treat_empty_val_as_null | BOOLEAN | [Optional] Specifies that empty fields become NULLs, rather than empty strings ('').
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
</tbody>
</table>

### Examples

1. Create a flex table for delimited data:

```sql
CREATE FLEX TABLE delim_flex ();
```

2. Use the `fdelimitedparser` to load some delimited data from STDIN, specifying a comma (,) column delimiter:

```sql
COPY delim_flex FROM STDIN parser fdelimitedparser (delimiter=',');
```

Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

```sql
>> deviceproduct, severity, deviceversion
>> ArcSight, High, 2.4.1
>> .
```

You can now query virtual columns in the `delim_flex` flex table:

```sql
SELECT deviceproduct, severity, deviceversion from delim_flex;
deviceproduct | severity | deviceversion
---------------|----------|--------------
ArcSight       | High     | 2.4.1
```

### See Also

- [FAVROPARSER](#)
- [FCEFPARSER](#)
FJSONPARSER

Parses and loads a JSON file. This file can contain either repeated JSON data objects (including nested maps), or an outer list of JSON elements. For a flex table, the parser stores the JSON data in a single-value VMap. For a hybrid or columnar table, the parser loads data directly in any table column with a column name that matches a key in the JSON source data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_maps</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.). Default value: true</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues. Default value: false</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>BOOLEAN</td>
<td>[Optional] Rejects any row containing a key without a value (reject_on_empty_key=true). Default value: false</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>BOOLEAN</td>
<td>[Optional] Omits any key from the load data that does not have a value (omit_empty_keys=true).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>record_terminator</td>
<td>STRING</td>
<td>[Optional] When set, any invalid JSON records are skipped and parsing continues with the next record. Records must be terminated uniformly. For example, if your input file has JSON records terminated by newline characters, specify fjsonparser(record_terminator=E'\n'). If any invalid JSON records exist, parsing continues after the next record_terminator. When you do not use a record_terminator, parsing ends at the first invalid JSON record. <strong>Default value:</strong> no default value</td>
</tr>
<tr>
<td>reject_on_materialized_type_error</td>
<td>BOOLEAN</td>
<td>[Optional] Rejects a data row that contains a materialized column value that cannot be coerced into a compatible data type (reject_on_materialized_type_error=true). <strong>Default value:</strong> false</td>
</tr>
<tr>
<td>start_point</td>
<td>CHAR</td>
<td>[Optional] Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the start_point value. The parser processes data after the first instance, and up to the second, ignoring any remaining data. <strong>Default value:</strong> none</td>
</tr>
<tr>
<td>start_point_occurrence</td>
<td>INTEGER</td>
<td>[Optional] Indicates the nth occurrence of the value you specify with start_point. Use in conjunction with start_point when load data has multiple start values and you know the occurrence at which to begin parsing. <strong>Default value:</strong> 1</td>
</tr>
<tr>
<td>suppress_nonalphanumeric_key_chars</td>
<td>BOOLEAN</td>
<td>[Optional] Suppresses non-alphanumeric characters in JSON key values. The parser replaces these characters with an underscore (_) when this parameter is true. <strong>Default value:</strong> false</td>
</tr>
</tbody>
</table>
key_separator | CHAR | [Optional] Specifies a non-default character for the parser to use when concatenating key names. Default value: '. '

Examples

Load JSON Data Without Optional Parameters

1. Create a flex table, super, with two columns, age and name:

```sql
create table super(age int, name varchar);
```

2. Enter values using the `fjsonparser()`, and query the results:

```sql
copy super from stdin parser fjsonparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> {"age": 5, "name": "Tim"}
>> {"age": 3}
>> {"name": "Fred"}
>> {"name": "Bob", "age": 10}
>> \. 
```

```sql
select * from super;
age | name
---------
  Fred 10  Bob
  5  Tim
  3 | ...
(4 rows)
```

For other examples, see Loading JSON Data.

See Also

- [FAVROPARSER](#)
- [FCEFPARSER](#)
- [FCSVPARSER](#)
- [FDELIMITEDPARSER](#)
FREGEXPARSER

Parses a regular expression, matching columns to the contents of the named regular expression groups.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>VARCHAR</td>
<td>Specifies the regular expression of data to match.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: Empty string (&quot;&quot;)</td>
</tr>
<tr>
<td>use_jit</td>
<td>BOOLEAN</td>
<td>[Optional] Indicates whether to use just-in-time compiling when parsing the regular expression.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>record_terminator</td>
<td>VARCHAR</td>
<td>[Optional] Specifies the character used to separate input records.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: \n</td>
</tr>
<tr>
<td>logline_column</td>
<td>VARCHAR</td>
<td>[Optional] Captures the destination column containing the full string that the regular expression matched.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: Empty string (&quot;&quot;)</td>
</tr>
</tbody>
</table>

Example

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.
This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, (<thread_id>(?:0x)?[0-9a-f]+).

'(?<text>.*)' (?:<thread_name>[A-Za-z ]+)?(?<thread_id>(?:0x)?[0-9a-f]+) 
\</?\<thread_id>[0-9a-f]+):(?:\[[<\?\<\<\<\?\<\text>\.*\).*\])*\) ?(\?\<text>\.*\)'

1. Create a flex table (vlog) to contain the results of a Vertica log file. For this example, we made a copy of a log file in the directory /home/dbadmin/data/vertica.log:

```sql
=> create flex table vlog1;
CREATE TABLE
```

2. Use the fregexparser with the sample regular expression to load data from the log file. Be sure to remove any line characters before using this expression shown here:

```sql
=> copy vlog1 from '/home/dbadmin/tempdat/KMvertica.log'
PARSER FREGEXPARSER('(?<text>.*)\</?\<thread_id>[0-9a-f]+):(?:\[[<\?\<\<\<\?\<\text>\.*\).*\])*\) ?(\?\<text>\.*\)'
\</?\<thread_id>[0-9a-f]+):(?:\[[<\?\<\<\<\?\<\text>\.*\]*)\) ?(\?\<text>\.*\)'
\</?\<thread_id>[0-9a-f]+):(?:\[[<\?\<\<\<\?\<\text>\.*\)])\) ?(\?\<text>\.*\)';
```

3. After successfully loading data, use the MAPTOSTRING() function with the table's __raw__ column. The four rows (limit 4) that the query returns are regular expression results of the KMVertica.log file, parsed with fregexparser. The output shows thread_id values with a preceding 0x or without:

```sql
=> select maptostring(__raw__) from vlog1 limit 4;
```

```
{
  "text": "[INFO] Log /home/dbadmin/VMart/v_vmart_node0001_catalog/vertica.log
opened; #2",
  "thread_id": "0x7f2157e287c0",
  "thread_name": "Main",
  "time": "2017-03-21 23:30:01.704"
}
{
  "text": "[INFO] Processing command line: /opt/vertica/bin/vertica -D
/home/dbadmin/VMart/v_vmart_node0001_catalog -C VMart -n v_vmart_node0001 -h
10.20.100.247 -p 5433 -P 4803 -Y ipv4",
  "thread_id": "0x7f2157e287c0",
}
```
See Also

- FDELIMITEDPAIRPARSER
- FDELIMITEDPARSER
- FJSONPARSER
Management Console (MC) is a user-friendly performance monitoring and management tool that provides a unified view of your Vertica database operations. Using a browser, you can create, import, manage, and monitor one or more databases and their associated clusters. You can also create and manage MC users. You can then map the MC users to a Vertica database and manage them through the MC interface.

The MC interface is provides tool tips for most of the MC operations. For an introduction to MC functionality, architecture, and security, see Management Console in Vertica Concepts.

To get started using MC, see Getting Started with MC.
Getting Started with MC

Use Management Console to monitor the performance of your Vertica clusters. This tool provides a graphical view of your Vertica database cluster, nodes, network status, and detailed monitoring charts and graphs.

MC allows you to:

- Create, import, and connect to Vertica database.
- Manage your Vertica database and clusters.
- Receive and view messages regarding the health and performance of your Vertica database and clusters.
- View diagnostics and support information for Management Console.
- Manage application and user settings for Management Console.

MC Installation Process

To install MC, complete these tasks:

1. Verify that you meet the requirements listed in Before You Install MC.
2. Follow the steps listed in Installing Management Console.
3. After you have installed MC, configure it according to the instructions in Configuring MC.

Connecting to MC

To connect to Management Console:
1. Open an HTML-5 compliant browser.

2. Enter the IP address or host name of the host on which you installed MC (or any cluster node if you installed Vertica first), followed by the MC port you assigned when you configured MC.

   For example, enter one of:

   https://00.00.00:5450/

   or

   https://hostname:5450/

3. When the MC logon dialog appears, enter your MC username and password and click Log in.

   Note: When MC users log in to the MC interface, MC checks their privileges on Vertica Data Collector (DC) tables on MC-monitored databases. Based on DC table privileges, along with the role assigned the MC user, each user's access to the MC's Overview, Activity and Node details pages could be limited. See About MC Privileges and Roles for more information.

   If you do not have an MC username/password, contact your MC administrator.

### Viewing the Home Page

After you connect to MC and sign in, the Home page displays. This page is the entry point to all Vertica database clusters and users managed by MC. Information on this page, as well as throughout the MC interface, will appear or be hidden, based on the permissions (access levels) of the user who is logged in. The following image is what an MC super administrator sees.
Tasks

Operations you can perform in Management Console are grouped into the following areas:

- **Provision.** Create new Vertica databases, or import existing ones to manage and monitor with MC. You can also import a Vertica cluster that resides in a Hadoop environment. See [Managing Database Clusters](#).

- **Manage.** View all the clusters and databases monitored by MC, stop and remove databases, and view details about databases and clusters. See [Viewing Cluster Infrastructure](#).

MC Tools

- **MC Settings.** Configure MC and user settings, as well as use the MC interface to install Vertica on a cluster of hosts. See [Managing MC Settings](#).

- **Message Center.** View, sort, and search database messages and optionally export messages to a file. See [Monitoring Database Messages in MC](#).
• **MC Diagnostics.** View and resolve MC-related issues, as well as browse Vertica agent and audit logs. See *Troubleshooting with MC Diagnostics.*

### Recent Databases

The Recent Databases section displays all databases that you created on or imported into MC. It lists each database name, its number of nodes, and two actions:

- **Fast Tasks.** Key tasks you can perform on that database using MC.
- **Go to Database.** View the Overview page, which displays a dynamic dashboard of your database's health and activity. Explore the tabs below your dashboard, which provide more ways to manage the database.

You can install and manage multiple databases with MC, but you can have only one database running on a single cluster at a time. UP databases appear in green and DOWN databases are red.

An empty space under Recent Databases means that you have not yet created or imported a database into the MC interface, or do not have permission to view any databases managed by MC.

### Managing MC Users, Roles and Privileges

If you are an administrator, you can use **MC Settings** to grant MC users privileges to one or more Vertica users. MC users are not the same as system (Linux) users. MC users are external to the database, and their information is stored on an internal database on the MC application or web server. See *About MC Users* for further details.

You can create MC users using either of two authentication techniques, LDAP or MC (internal). See *Creating an MC User.* After you create the MC users, you can manage them from **MC Settings** page. Refer to *Managing MC Users.*

To control the level of access for the MC Users, you can grant them privileges (through roles) from the **MC Settings** page. MC supports two groups of privileges:

- **MC Configuration Privileges**
- **MC Database Privileges**
The MC super account is the default user. The super user needs to create all other MC users. Refer to About MC Privileges and Roles for further information on MC roles.

For further details about MC Users, Privileges and Roles, see Managing Users And Privileges.

Creating a Cluster Using MC - Process Flow

After you install and configure MC, you can use it to create a Vertica cluster on hosts where Vertica software is not installed. Complete the following tasks:

1. Prepare the Hosts - Prepare each host that will become a node in the cluster.

2. Create a Private Key File - MC needs password-less SSH to connect to hosts and install Vertica software. Create a private key to enable MC access to the hosts.

3. Use the MC Cluster Installation Wizard - Use the wizard to install a Vertica cluster on hosts that do not have Vertica software already installed on them.

4. Validate Hosts and Create the Cluster - Host validation is the process where the MC runs tests against each host in a proposed cluster. You must validate hosts before the MC can install Vertica on each host.

After you successfully create a cluster using MC, see Create a Database on a Cluster.

Creating a Database Using MC

Follow these steps to create a database on an existing cluster using Management Console.

Important: This task assumes that you have already installed Management Console and are familiar with its concepts and layout. For more information, see Using Management Console, Management Console in the Concepts Guide, and Installing and Configuring Management Console in the Installation Guide.

1. Connect to Management Console, and log in.

2. On the Home page, under the Manage Information pane, click Existing Infrastructure.
3. On the following screen, click to select an existing cluster. When a dialog box identifying that cluster appears, click Create Database. The database creation wizard starts.

4. Follow the steps in the wizard to successfully create a database.

See Also

- Create an Empty Database Using MC
- Importing an Existing Database Into MC
- Creating a Cluster Using MC - Process Flow
Provisioning Databases Using MC

Management Console allows all users to create, import, and connect to Vertica databases using the MC Provision Databases tab.

- Import cluster or database using IP discovery
- Create a new cluster
- Import and Monitor in a Hadoop Environment

Fast Tasks

The Fast Tasks page offers a few important tasks to get you started managing your database through MC.

On the MC home page in the Recent Databases section, click Fast Tasks in the Actions column of any available database.

The Fast Tasks page also appears when you click Get Started after creating a database using Vertica Management Console with Provisioning.

The Fast Tasks page provides the following options:

- Manage and View Your Vertica Database. See the monitoring dashboard for your database. Explore the tabs below your dashboard, which provide more ways to manage the database.

- Connect to your Vertica Database Using SQL. Use Management Console to run SQL queries on your database from within your browser.

- Load Your Data into Vertica Database. The Data Load Activity page allows you to monitor and perform data loading jobs. To load data from an S3 bucket into pre-existing tables in your database, select the Instance tab and click New S3 Data Load

If you installed Management Console with Provisioning through the AWS Marketplace, you had the option to install a database pre-loaded with example clickstream analytic data. If you did so, the Fast Tasks page lists two additional links:

- Play with Example Data in Your Vertica Database (PDF). This PDF guide provides instructions and example SQL queries you can use in Vertica to analyze the example data.
Vertica Workbook for Tableau (PDF). This PDF guide provides instructions on how to analyze the example data pre-loaded into Vertica, using a Tableau dashboard.

Monitoring Existing Infrastructure Using MC

Use the Management Console Existing Infrastructure tab to monitor the health of your Vertica databases and clusters. Click the cluster of interest to view the health of the nodes in that cluster and the key information associated with the cluster such as:

- Vertica version
- Number of hosts
- CPU type
- Last updated date
- Node list.

You can also zoom in and out for better view of this page.

Click the database which you want to monitor to go to its Overview page:
Tip: You can also view the Overview page by clicking on the desired database on the MC Home Page.

You can perform the following tasks from the Overview page:

- View Quick Stats to get instant alerts and information about your cluster's status.
- View Status Summary that provides a general overview of the status of your cluster (as shown in preceding figure).
- Analyze System Health using a comprehensive summary of your system resource usage and node information, with configurable statistics that allow you to specify acceptable ranges of resource usage.
- Use Query Synopsis to monitor system query activity and resource pool usage.

Additionally, you can perform the following tasks from the Overview page:

- Monitoring Cluster Nodes
- Monitoring Cluster CPU/Memory
- Monitoring Cluster Performance

Monitoring System Resources

On the main window, you can click the database, and navigate to the MC Activity tab to monitor system resources such as:

- Queries
- Internal Sessions
- User Sessions
- Memory Usage
- System Bottlenecks
- User Query Phases
- Table Treemap
Query Monitoring

Resource Pool Monitoring

Monitoring Node and MC User Activity

You can use the MC Manage page to monitor node activity. When you click the node you want to investigate, the Node Detail page opens and provides:

- Summary information for the node
- Resources consumed by the node for last three hours

You can also browse and export log-level data from AgentTools and Vertica log files. MC retains a maximum of 2000 log records. See Monitoring Node Activity for further details.

Use MC Diagnostics tab and navigate to Audit Log page to manage MC User activity. See Monitoring MC User Activity Using Audit Log.

Monitoring messages in Databases managed by MC

You can view critical database related messages from MC Message Center. The MC Message Center reports on several critical database-related conditions using a color code to indicate the message severity. See Monitoring Database Messages in MC for further details.

You can also search and sort database messages, mark messages read or unread and delete them. You can filter messages by message type, and export messages. Refer to Searching Database Messages Managed by MC and Exporting MC-managed Database Messages and Logs.

Monitoring and Configuring Resource Pools

Use the MC Activity page to monitor resource pools. Select the resource pool you want to monitor. MC displays the following charts for the selected pool:

- Resource Usages in Pool
- Memory Usage in Node
• Average Query Execution and Query Time in Pool

• Resource Rejections in Pool

If you are a database administrator, you can click the database you want on the main window. You can then use the MC Settings tab to view and edit the resource pool parameters. Only the database administrator can monitor and configure the resource pools in Management Console. See [Monitoring Resource Pools](#) for further information.

### Running Database Designer Using MC

You can use Database Designer to create a comprehensive design, which allows you to create new projections for all tables in your database.

Additionally, you can use Database Designer to create an incremental design. An incremental design creates projections for all tables referenced in the queries you supply.

To run Database Designer using MC, follow the steps listed at [Running Database Designer with Management Console](#).

### Managing Queries Using MC

Management Console allows you to view the query plan of an active query or a manually entered query specified by the user.

1. On the [MC Home Page](#), click the database you want to view the Overview page.

2. Select the Activity tab to view the query activity.

3. Click the Explain tab to access the query plan.

See [Managing Queries in MC](#) and [Accessing Query Plans in Management Console](#) for further information.

Management Console provides two options for viewing the query plan: Path Information and Tree Path. For details on each, refer [Query Plan View Options](#).

Additionally, you can also [Viewing Projection and Column Metadata](#) using the MC Explain tab.
See Also

- Expanding and Collapsing Query Paths
- Clearing Query Data

**Profiling Queries Using MC**

Management Console allows you to view profile data for a query.

- On the **MC Home Page**, click the database to view the Overview page.
- Click the Explain tab to perform tasks related to profiling a query.

See [Viewing Profile Data in MC](#) for further details.

On the Explain tab, you can view the following profile data using MC:

- **Query Phase Duration**
- **Projection Metadata**
- **Execution Events**
- **Optimizer Events**
- **Profile Metrics**

You can use any of the four different formats to view the profile data:

- Path Information view
- Query Drilldown view
- Tree Path view
- Profile Analysis view

See [Viewing Different Profile Outputs](#) for detailed explanation of each view.

Additionally, Management Console supports different color codes for viewing the progress of profiling a query. For an explanation of these color codes, see [Monitoring Profiling Progress](#).
See Also

- Viewing Profile Data in MC

Managing Client Connections

Each client session to MC uses a connection from MaxClientSessions, a database configuration parameter. This parameter determines the maximum number of sessions that can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for MaxClientSessions on an MC-monitored database to account for extra sessions. See Managing Sessions for details.

Managing MC Settings

The MC Settings page allows you to configure properties specific to Management Console. You can:

- Change MC and agent default port assignments.

- Enable and disable username and password auto-complete at MC login. (After disabling, clear your browser's cache.)

- Control the following monitoring settings in MC:
  - Enable checks and set alert thresholds for spread retransmit rate. This setting is disabled by default. The recommended alert threshold for spread retransmit rate is 10%.
  - Set alert thresholds for free MC disk space checks. The recommended alert threshold is 500 MB.
  - Exclude MC queries from activity charts.
  - Set refresh intervals for MC charts and pages.

- Upload a new SSL certificate.
- Use LDAP for user authentication.
- Create new MC users and, with their user credentials, map them to an database managed by MC on the Vertica server. See *Creating an MC User* and *Managing MC Users*.
- View your version of Vertica or upload a new Vertica binary file.
- Customize the look and feel of MC with themes. See *Customizing Look and Feel*.
- Configure MC to use an alternative data source to monitor your database. See *Monitoring External Data Sources in Management Console*.
- Enable MC to send email alerts. See *Set Up Email*.
- Configure *Extended Monitoring*, which allows you to monitor more long-term data in MC:
  - Set up an external storage database for Extended Monitoring. See *Managing the Storage Database*.
  - Enable or disable Extended Monitoring on your databases. See *Managing Extended Monitoring on a Database*.

### Modifying Database-Specific Settings

To inspect or modify settings related to a database managed by MC, go to the Existing Infrastructure page. On this page, select a running database to see its Overview page. From the bottom of the Overview page, click the Settings tab to make modifications to database-specific settings.

### Backing Up MC

Before you upgrade MC, Vertica recommends that you back up your MC metadata (configuration and user settings). Use a storage location external to the server on which you installed MC.

1. On the target server (where you want to store MC metadata), log in as root or a user with sudo privileges.
2. Create a backup directory as in following example:
3. Copy the /opt/vconsole directory to the new backup folder:

```
# cp -r /opt/vconsole /backups/mc/mc-backup-20130425
```

# Upgrading And Uninstalling MC

To upgrade or uninstall MC refer to Installing Vertica

- To upgrade MC, follow the steps listed on Upgrading Management Console.
- To uninstall MC, refer Uninstalling Management Console.
Managing Users And Privileges

A Management Console administrator can grant MC users access to one or more Vertica databases through the MC interface. In this section, we discuss about MC Users and their privileges.

- About MC Users
- About MC Privileges and Roles
About MC Users

Unlike database users, which you create on the Vertica database and then grant privileges and roles through SQL statements, you create MC users on the Management Console interface. MC users are external to the database. Their information is stored on an internal database on the MC application/web server. Their access to both MC and to databases managed by MC is controlled by groups of privileges (also referred to as access levels). MC users are not system (Linux) users; they are entries in the MC internal database.

Permission Group Types

There are two types of permission groups on MC, those that apply to MC configuration and those that apply to database access:

- **MC configuration** privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create and import Vertica databases, restart MC, create a Vertica cluster through the MC interface, and create and manage MC users.

- **MC database** privileges are made up of roles that control what users can see or do on a Vertica database monitored by MC, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

If you are using MC, you might want to allow one or more users in your organization to configure and manage MC, and you might want other users to have database access only. You can meet these requirements by creating MC users and granting them a role from each privileges group. See Creating an MC User for details.

MC User Types

There are five types of role-based users on MC:

- The default superuser administrator (Linux account) who gets created when you install and configure MC and oversees all of MC. See SUPER Role (mc).

- Users who can configure all aspects of MC and control all databases managed by MC. See ADMIN Role (mc).
- Users who can configure MC user settings and monitor all databases managed by MC. See MANAGER Role (MC).

- Users who can configure some aspects of MC user settings and monitor all databases managed by MC. See IT Role (mc).

- Users who cannot configure MC and have access to one or more databases managed by MC. See NONE Role (mc).

You create users and grant them privileges (through roles) on the MC Settings page in the User management tab.

Creating Users and Choosing an Authentication Method

You create users and grant them privileges (through roles) on the MC Settings page. You can also choose how to authenticate their access to MC.

- To add users who are authenticated against the MC, click User Management

- To add users who are authenticated through your organization's LDAP repository, click Authentication

MC supports only one method for authentication, so if you choose MC, all MC users will be authenticated using their MC login credentials.

Default MC Users

The MC super account is the only default user. The super or another MC administrator must create all other MC users.

See Also

- Management Console
- About MC Privileges and Roles
- Granting Database Access to MC Users
Creating an MC User

MC provides two authentication schemes for MC users: LDAP or MC (internal). Which method you choose will be the method MC uses to authenticate all MC users. It is not possible to authenticate some MC users against LDAP and other MC users against credentials in the database through MC.

- **MC (internal) authentication.** Internal user authorization is specific to MC itself. You create a user with a username and password combination. This method stores MC user information in an internal database on the MC application/web server, and encrypts passwords. Note that these MC users are not system (Linux) users; they are entries in the MC’s internal database.

- **LDAP authentication.** All MC users—except for the MC super administrator, which is a Linux account—are authenticated based on search criteria against your organization’s LDAP repository. MC uses information from LDAP for authentication purposes only and does not modify LDAP information. Also, MC does not store LDAP passwords but passes them to the LDAP server for authentication.

Instructions for creating new MC users are in this topic.

- If you chose MC authentication, follow the instructions under Create a New User Authenticated by MC.

- If you chose LDAP authentication, follow the instructions under Create a New User from LDAP.

See About MC Users and LDAP Authentication for more information.

Prerequisites

Before you create an MC user, you already:

- Created a database directly on the server or through the MC interface, or you imported an existing database cluster into the MC interface. See Managing Database Clusters.

- Created a database user account (source user) on the server, which has the privileges and/or roles you want to map to the new (target) MC user. See Creating a Database User.

- Know what MC privileges you want to grant the new MC user. See About MC Privileges and Roles.
If you have not yet met the first two above prerequisites, you can still create new MC users; you just won't be able to map them to a database until after the database and target database user exist. To grant MC users database access later, see Granting Database Access to MC Users.

Create a New User Authenticated by MC

1. Sign in to Management Console as an administrator and navigate to MC Settings > User management.

2. Click Add.

3. Enter the MC username.

   **Note:** It is not necessary to give the MC user the exact same name as the database user account you'll map the MC user to in Step 7. What matters is that the source database user has privileges and/or roles similar to the database role you want to grant the MC user. The most likely scenario is that you will map multiple MC users to a single database user account.

4. Let MC generate a password or create one by clicking Edit password. If LDAP has been configured, the MC password field will not appear.

5. Optionally enter the user's e-mail address.

6. Select an MC configuration permissions level. See MC Configuration Privileges.

7. Next to the DB access levels section, click Add to grant this user database permissions.

   i. Choose a database. Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).

   ii. Database username. Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.

   iii. Database password. Enter the password to the database user account (not this username's password).

   iv. Restricted access. Chose a database level (**ADMIN**, **IT**, or **USER**) for this user.

   v. Click OK to close the Add permissions dialog box.
8. Leave the user's Status as enabled (the default). If you need to prevent this user from accessing MC, select disabled.

9. Click Add User to finish.

Create a New LDAP-authenticated User

When you add a user from LDAP on the MC interface, options on the Add a new user dialog box are slightly different from when you create users without LDAP authentication. Because passwords are store externally (LDAP server) the password field does not appear. An MC administrator can override the default LDAP search string if the user is found in another branch of the tree. The Add user field is pre-populated with the default search path entered when LDAP was configured.

1. Sign in to Management Console and navigate to MC Settings > User management.

2. Click Add and provide the following information:
   a. LDAP user name.
   b. LDAP search string.
   c. User attribute, and click Verify user.
   d. User's email address.
   e. MC configuration role. NONE is the default. See MC Configuration Privileges for details.
   f. Database access level. See MC Database Privileges for details.
   g. Accept or change the default user's Status (enabled).

3. Click Add user.

If you encounter issues when creating new users from LDAP, you'll need to contact your organization's IT department.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your
organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.

See Also

- Configuring MC
- About MC Users
- About MC Privileges and Roles
- Granting Database Access to MC Users
- Creating a Database User

Managing MC Users

You manage MC users through the following pages on the Management Console interface:

- MC Settings > User management
- MC Settings > Resource access

Who Manages Users

The MC superuser administrator (SUPER Role (mc)) and users granted ADMIN Role (mc) manage all aspects of users, including their access to MC and to MC-managed databases.

Users granted IT Role (mc) can enable and disable user accounts.

See About MC Users and About MC Privileges and Roles for more information.

Editing an MC user's information follows the same steps as creating a new user, except the user's information will be pre-populated, which you then edit and save.

The only user account you cannot alter or remove from the MC interface is the MC super account.
What Kind of User Information You Can Manage

You can change the following user properties:

- MC password

- Email address. This field is optional. If the user is authenticated against LDAP, the email field is pre-populated with that user's email address if one exists.

- **MC Configuration Privileges** role

- **MC Database Privileges** role

You can also change a user's status (enable/disable access to MC) and delete users.

About User Names

After you create and save a user, you cannot change that user's MC user name, but you can delete the user account and create a new user account under a new name. The only thing you lose by deleting a user account is its audit activity, but MC immediately resumes logging activity under the user's new account.
About MC Privileges and Roles

As introduced in About MC Users, you control user access to Management Console through groups of privileges (also referred to as access levels) that fall into two types, those that apply to MC configuration, and those that apply to MC-managed Vertica databases.

MC Permission Groups

- **MC configuration** privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create and import Vertica databases, restart MC, create a Vertica cluster through the MC interface, and create and manage MC users.

- **MC database** privileges are made up of roles that control what users can see or do on a Vertica database monitored by MC, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

Note: When you grant an MC user a database role, that user inherits the privileges assigned to the database user account to which the MC user is mapped. For maximum access, use the dbadmin username and password.

MC database privileges cannot alter or override the Vertica database user's privileges and roles.

See Also

- About MC Users
- Creating an MC User
- Managing MC Users, Roles and Privileges
- MC Database Privileges
- Creating an MC User
- Granting Database Access to MC Users
MC Configuration Privileges

When you create an MC user, you assign them an MC configuration access level (role). MC roles control a user's ability to create users and manage MC settings on the MC interface.

In addition to an MC role, users also have a database role that controls their database-specific privileges. See MC Database Privileges.

MC Roles and Privileges You Can Assign Users

You can assign a user one of the following MC access levels:

- **ADMIN Role (mc)**—Full access to all MC functionality.
- **MANAGER Role (MC)**—Access to MC user management functionality. Access to non-database MC alerts.
- **IT Role (mc)**—Limited access to MC user management functionality. Access to MC log and to non-database MC alerts.
- **NONE Role (mc)**—Database access only, to the databases an administrator assigns to this user.

You grant MC configuration privileges at the same time you create the user's account, on the User Management tab of the MC Settings page. You can change MC access levels using this page. See Creating an MC User for details.

You can also use the User Management tab to grant users access to one or more databases managed by MC. See MC Database Privileges for details.

MC Configuration Privileges By User Role

You grant the following configuration privileges by MC role.

<table>
<thead>
<tr>
<th>MC access privileges</th>
<th>ADMIN</th>
<th>MANAGER</th>
<th>IT</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure MC settings:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Configure storage locations and ports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Upload new SSL certificates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configure MC settings:</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MC access privileges</strong></td>
<td><strong>ADMIN</strong></td>
<td><strong>MANAGER</strong></td>
<td><strong>IT</strong></td>
<td><strong>NONE</strong></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Manage LDAP authentication</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Vertica installation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change MC theme</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map to an external data source</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configure user settings:</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add, edit, delete users</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add, change, delete user permissions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map users to one or more databases</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configure user settings:</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Enable or disable user access to MC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset user passwords</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor user activity on MC using audit log</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create and manage databases and clusters:</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create a new database or import an existing one</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create a new cluster or import an existing one</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove databases and clusters from MC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset MC to its original, preconfigured state</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restart Management Console</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View full list of databases monitored by MC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>View MC log</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>View non-database MC alerts</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
See Also

- About MC Users
- About MC Privileges and Roles
- Managing MC Users, Roles and Privileges
- MC Database Privileges
- Creating an MC User
- Granting Database Access to MC Users

SUPER Role (mc)

The default superuser administrator, called Super on the MC UI, is a Linux user account that gets created when you install and configure MC. During the configuration process, you can assign the Super any name you like; it need not be dbadmin.

The MC SUPER role, a superset of the ADMIN Role (mc), has the following privileges:

- Oversees the entire Management Console, including all MC-managed database clusters

  Note: This user inherits the privileges/roles of the user name supplied when importing a Vertica database into MC. Vertica recommends that you use the database administrator's credentials.

- Creates the first MC user accounts and assigns them an MC configuration role

- Grants MC users access to one or more MC-managed Vertica databases by assigning MC Database Privileges to each user

The MC super administrator account is unique. Unlike other MC users you create, including other MC administrators, the MC super account cannot be altered or dropped, and you cannot grant the SUPER role to other MC users. The only property you can change for the MC super is the password. Otherwise the SUPER role has the same privileges on MC as the ADMIN Role (mc).

On MC-managed Vertica databases, SUPER has the same privileges as ADMIN Role (db).

The MC super account does not exist within the LDAP server. This account is also different from the special dbadmin account that gets created during a Vertica installation, whose privileges
are governed by the **DBADMIN Role**. The Vertica-created dbadmin is a Linux account that owns the database catalog and storage locations and can bypass database authorization rules, such as creating or dropping schemas, roles, and users. The MC super does not have the same privileges as dbadmin.

**See Also**

- Configuring MC
- About MC Privileges and Roles
- Creating an MC User
- Granting Database Access to MC Users
- Managing MC Users

**ADMIN Role (mc)**

This user account is the user who can perform all administrative operations on Management Console, including configure and restart the MC process and add, change, and remove all user accounts. By default, MC administrators inherit the database privileges of the main database user account used to set up the database on the MC interface. Therefore, MC administrators have access to all MC-managed databases. Grant the ADMIN role to users you want to be MC administrators.

The difference between this ADMIN user and the default Linux account, the MC SUPER role, is you cannot alter or delete the MC SUPER account, and you can't grant the SUPER role to any other MC users. You can, however, change the access level for other MC administrators, and you can delete this user's accounts from the MC interface.

The following list highlights privileges granted to the ADMIN role:

- Modify MC settings, such as storage locations and ports, restart the MC process, and reset MC to its original, unconfigured state
- Audit license activity and install/upgrade a Vertica license
- Upload a new SSL certificate
- Use LDAP for user authentication
- View the MC log, alerts and messages
- Add new users and map them to one or more Vertica databases by granting an MC database-level role
- Select a database and add multiple users at once
- Manage user roles and their access to MC
- Remove users from the MC
- Monitor user activity on the MC interface
- Stop and start any MC-managed database
- Create new databases/clusters and and import existing databases/clusters into MC
- Remove databases/clusters from the MC interface
- View all databases/clusters imported into MC

About the MC Database Administrator Role

There is also an MC database administrator (ADMIN) role that controls a user's access to MC-managed databases. The two ADMIN roles are similar, but they are not the same, and you do not need to grant users with the ADMIN (mc) role an ADMIN (db) role because MC ADMIN users automatically inherit all database privileges of the main database user account that was created on or imported into MC.

The following table summarizes the primary difference between the two ADMIN roles, but see ADMIN Role (db) for details specific to MC-managed database administrators.

<table>
<thead>
<tr>
<th>MC configuration ADMIN role</th>
<th>MC database ADMIN role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform all administrative operations on the MC itself, including restarting the MC process. Privileges extend to monitoring all MC-created and imported databases but anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements.</td>
<td>Perform database-specific activities, such as stop and start the database, and monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC.</td>
</tr>
</tbody>
</table>
MANAGER Role (MC)

Users assigned the Manager role can configure user settings in MC. The Manager role allows full access to the User Management tab in MC Settings. Managers can also view a full list of databases monitored by MC on the Home page, view the MC log, and see non-database MC alerts.

The Manager role has similar configuration privileges to the IT configuration role. Unlike IT users, Managers can also create, edit, and delete users in User Settings.

Managers can:

- Add, edit, delete users
- Add, change, delete user permissions
- Map users to one or more databases
- Enable or disable user access to MC
- Reset user passwords
- View the full list of databases monitored by MC on the MC Home page
- View the MC log
- View non-database MC alerts

IT Role (mc)

MC IT users can monitor all MC-managed databases, view MC-level (non database) messages, logs, and alerts, disable or enable user access to MC, and reset non-LDAP user passwords. You can also assign MC IT users specific database privileges, which you do by mapping IT users to a user on a database. In this way, the MC IT user inherits the privileges assigned to the database user to which he/she is mapped.

About the MC IT (database) Role

There is also an IT database administrator (IT) role that controls a user's access to MC-managed databases. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The database mapping is not required, but it gives the IT user wider privileges.
The two IT roles are similar, but they are not the same. The following table summarizes the primary difference between them, but see IT Role (db) for details.

<table>
<thead>
<tr>
<th>MC configuration IT role</th>
<th>MC database IT role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor MC-managed database, view non-database messages, and manage user access</td>
<td>Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings. Can also be mapped to one or more Vertica databases.</td>
</tr>
</tbody>
</table>

NONE Role (mc)

The default role for all newly-created users on MC is NONE, which prevents users granted this role from configuring the MC. When you create MC users with the NONE role, you grant them an MC database-level role. This assignment maps the MC user to a user account on a specific database and specifies that the NONE user inherits the database user’s privileges to which he or she is mapped.

Which database-level role you grant this user with NONE privileges—whether ADMIN (db) or IT (db) or USER (db)—depends on the level of access you want the user to have on the MC-managed database. Database roles have no impact on the ADMIN and IT roles at the MC configuration level.

MC Database Privileges

When you create MC users, you first assign them MC configuration privileges, which controls what they can do on the MC itself. In the same user-creation operation, you grant access to one or more MC-managed databases. MC database access does not give the MC user privileges directly on Vertica; it provides MC users varying levels of access to assigned database functionality through the MC interface.

Assign users an MC database level through one of the following roles:

- **ADMIN Role (db)**—Full access to all databases managed by MC. Actual privileges ADMINs inherit depend on the database user account used to create or import the Vertica database into the MC interface.
• **Associate Role (Database)**—Full access to all databases managed by MC. Cannot start, stop, or drop a database. Actual privileges that Associates receive depend on those defined for the database user account to which the Associate user is mapped.

• **IT Role (db)**—Can start and stop a database but cannot remove it from the MC interface or drop it.

• **USER Role (db)**—Can view database information through the database Overview and Activities pages but is restricted from viewing more detailed data.

### Mapping MC Users to Database to Avoid Conflicts

When you assign an MC database level to an MC user, map the MC user account to a database user account to:

- Inherit the privileges assigned to that database user
- Prevent the MC user from doing or seeing anything not allowed by the privileges for the user account on the server database

Privileges assigned to the database user supersede privileges of the MC user if there is a conflict, such as stopping a database. When the MC user logs into MC using an MC user name and password, Vertica compares privileges for database-related activities to the privileges on the database account to which you mapped the MC user. Vertica allows operations in MC to the user only when that user has both MC privileges and corresponding database privileges.

| Tip: As a best practice, you should identify, in advance, the appropriate Vertica database user account that has privileges or roles similar to one of the MC database roles. |

See [Creating an MC User](#) for more information.

### MC Database Privileges By Role

The following table summarizes MC database-level privileges by user role. The table shows the default privileges each role has. Operations marked "database user privilege" are dependent on the privileges of the Vertica database user account to which the MC user is mapped.

<table>
<thead>
<tr>
<th>Default database-level privileges</th>
<th>ADMIN</th>
<th>ASSOCIATE</th>
<th>IT</th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>View database Overview</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Default database-level privileges</td>
<td>ADMIN</td>
<td>ASSOCIATE</td>
<td>IT</td>
<td>USER</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------</td>
<td>-----------</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>page</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View database messages</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Delete messages and mark read/unread</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Audit and install Vertica licenses</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
<td></td>
<td></td>
</tr>
<tr>
<td>View database Activity page:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Queries chart</td>
<td>Yes</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
</tr>
<tr>
<td>• Internal Sessions chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• User Sessions chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• System Bottlenecks chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• User Query Phases chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View database Activity page:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Queries chart &gt; Detail page</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Table Treemap chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Query Monitoring chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Resource Pools Monitoring chart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start a database</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Default database-level privileges

<table>
<thead>
<tr>
<th>Default database-level privileges</th>
<th>ADMIN</th>
<th>ASSOCIATE</th>
<th>IT</th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebalance, stop, or drop databases</td>
<td>Database user privilege</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View Manage page</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>View node details</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Replace, add, or remove nodes</td>
<td>Database user privilege</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start/stop a node</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View database Settings page</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Modify database Settings page</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
<td></td>
<td></td>
</tr>
<tr>
<td>View Database Designer</td>
<td>Database user privilege</td>
<td>Database user privilege</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ADMIN Role (db)**

ADMIN is a superuser with full privileges to monitor MC-managed database activity and messages. Other database privileges (such as stop or drop the database) are governed by the user account on the Vertica database that this ADMIN (db) user is mapped to. ADMIN is the most permissive role and is a superset of privileges granted to the Associate Role (Database), IT, and USER roles.

The ADMIN user has the following database privileges by default:

- View the database Overview page
- View and delete database messages
- Mark messages read or unread
- Start the database
- View the Manage page
- View node details
- Start or stop a node
- View database settings
- View the following database Activity page charts:
  - Queries
  - Internal Sessions
  - User Sessions
  - System Bottlenecks
  - User Query Phases

The following database operations depend on the database user’s role that you mapped this ADMIN user to:

- Install or audit a license
- Rebalance, stop, or drop databases
- Add, replace, or remove nodes
- Manage database settings
- View Database Designer
- View additional information on the database Activity page:
  - Detailed information in the Queries chart Detail page
  - Table Treemap chart
  - Query Monitoring chart
  - Resource Pools Monitoring chart

Note: Database access granted through Management Console never overrides roles granted on a specific Vertica database.
About the ADMIN (MC configuration) Role

There is also an MC configuration administrator role that defines what the user can change on the MC itself. The two ADMIN roles are not the same. Unlike the MC configuration role of ADMIN, which can manage all MC users and all databases imported into the UI, the MC database ADMIN role has privileges only on the databases you map this user to. The following table summarizes the primary difference between them. See ADMIN Role (mc) for additional details.

<table>
<thead>
<tr>
<th>MC database ADMIN role</th>
<th>MC configuration ADMIN role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform database-specific activities, such as stop and start the database, and monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC.</td>
<td>Perform all administrative operations on the MC itself, including restarting the MC process. Privileges extend to monitoring all databases created and imported by MC. Anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements.</td>
</tr>
</tbody>
</table>

Associate Role (Database)

The Associate role is an MC database access role. It is similar to the Admin role. It has privileges to monitor activity and messages on databases managed by MC. Unlike Admin users, Associate users cannot start, stop, or drop the database. The Associate user role is mapped to a user account on the database. This mapped user role determines what other database privileges the Associate role has (such as modifying settings, installing licenses, and viewing the database designer).

The Associate user has the following database privileges by default:

- View the database Overview page
- View and delete database messages
- Mark messages read or unread
- View the Manage page
- View node details
- View database settings
The following database operations depend on the database user's role that you mapped this Associate user to:

- Install or audit a license
- Manage database settings
- View Database Designer
- View the database Activity page

**Note:** Database access granted through Management Console never overrides roles granted on a specific Vertica database.

**IT Role (db)**

IT can view most details about an MC-managed database, such as messages (and mark them read/unread), the database overall health and activity/resources, cluster and node state, and MC settings. You grant and manage user role assignments through the MC Settings > User management page on the MC.

**About the IT (MC configuration) Role**

There is also an IT role at the MC configuration access level. The two IT roles are similar, but they are not the same. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The following table summarizes the primary difference between them, but see **IT Role (mc)** for additional details.

<table>
<thead>
<tr>
<th>MC database IT</th>
<th>MC configuration IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings.</td>
<td>Monitor MC-managed database, view non-database messages, and manage user access.</td>
</tr>
</tbody>
</table>

**USER Role (db)**

USER has limited database privileges, such as viewing database cluster health, activity/resources, and messages. MC users granted the USER database role might have higher
levels of permission on the MC itself, such as the IT Role (mc). Alternatively, USER users might have no (NONE) privileges to configure MC. How you combine the two levels is up to you.

**Granting Database Access to MC Users**

If you did not grant an MC user a database-level role when you created the user account, you can do so in the User Management tab in MC Settings.

Granting the user an MC database-level role associates the MC user with a database user's privileges and ensures that the MC user cannot do or see anything not allowed by the privileges set up for the user account on the server database. When that MC user logs in to MC, his or her MC privileges for database-related activities are compared to that user's privileges on the database itself. Only when the user has both MC privileges and corresponding database privileges will the operations be exposed in the MC interface.

**Prerequisites**

Before you grant database access to an MC user, see the prerequisites in Creating an MC User.

**Grant a Database-Level Role to an MC user**

1. Log in to Management Console as an administrator and navigate to MC Settings > User management.
2. Select an MC user and click Edit.
3. Verify the MC Configuration Privileges are what you want them to be. NONE is the default.
4. Next to the DB access levels section, click Add and provide the following database access credentials:
   i. Choose a database. Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).
   ii. Database username. Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.
   iii. Database password. Enter the password to the database user account (not this username's password).
iv. Restricted access. Chose a database level (ADMIN, IT, or USER) for this user.

v. Click OK to close the Add permissions dialog box.

5. Optionally change the user's Status (enabled is the default).

6. Click Save.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.
Managing Client Connections

Each client session to MC uses a connection from MaxClientSessions, a database configuration parameter. This parameter determines the maximum number of sessions that can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for MaxClientSessions on an MC-monitored database to account for extra sessions. See Managing Sessions for details.
Managing Database Clusters

MC allows you to monitor multiple databases on one or more clusters at once. MC administrators can see and manage all databases and clusters monitored by MC, while non-administrative MC users see only databases on which they have been assigned the appropriate access levels.

Depending on your access level, the database and cluster-related management operations you can perform through MC include:

- Create an empty database in an existing cluster.
- Create a cluster and database on AWS.
- Import an existing database/cluster into the MC interface.
- Start the database, unless it is already running (green).
- Stop the database, if no users are connected.
- Remove the database from the MC interface.

Note: Remove does not drop the database. A Remove operation leaves it in the cluster, hidden from the UI. To add the database back to the MC interface, import it using the IP address of any cluster node. A Remove operation also stops metrics gathering on that database, but statistics gathering automatically resumes after you re-import.

- Drop the database after you ensure no users are connected. Drop is a permanent action that drops the database from the cluster.

In databases and clusters created using Management Console with Provisioning, AWS instance operations are available from MC, like stop, start, reboot and terminate. These AWS operations can be performed on the whole cluster or one instance at a time. See the sections Managing a Cluster on AWS Resources and Managing AWS Instances sections in Viewing and Managing Your Cluster.

Viewing Cluster Infrastructure

For a summary of all databases and clusters currently monitored by MC, click View Your Infrastructure on the MC Home page.
The first tab on the Infrastructure page, Database and Cluster View, is overview of the infrastructure of all the clusters and databases currently monitored by MC.

Three rows are displayed: Infrastructure, Clusters, and Databases.

- **Infrastructure.** This is type of environment (on-premises, AWS, or Apache Hadoop) on which your clusters reside.
- **Clusters.** Click a cluster to see its full details. From the dialog that opens, you can:
  - Add the cluster's master API key
  - View or Manage the cluster page
  - Remove the cluster from MC monitoring
  - Create a new database on the cluster (if all other databases on the cluster are stopped)
- **Databases.** A numbered badge on the top right displays the number of highest-priority messages from that database are in your inbox. If a handshake icon (🔗) displays next to "Type," that indicates the database is running in Eon Mode Beta.

Click any database for more details. From the dialog that opens, you can:
- View the database's Overview page
- Stop or start the database
- Drop the database (if the database is stopped)
- Remove the database from MC monitoring

In the screen capture below, MC is monitoring two different clusters that both reside on an AWS environment. One database is running on each cluster. The DemoDB database, displayed on the left, has a handshake icon next to its Type label that indicates it is running in Eon Mode Beta. The VMart database on the 3-node cluster, displayed on the right, is running in Enterprise Mode.
Viewing and Managing Your Cluster

The Cluster page shows a node-based visualization of your cluster. On this page you will see the cluster's host addresses, the installed version of Vertica running, and a list of the databases on the cluster that MC is currently monitoring.
From this page, you can also create a new empty database on the cluster, or import any existing databases MC discovers on the cluster.

The Cluster page shows additional AWS cluster and instance management options only if you created your cluster on AWS resources using the Cluster Creation wizard in MC (available using Management Console with Provisioning). See the section below, Managing a Cluster Created on AWS with the Cluster Creation Wizard.

Management options are not available for clusters created without using the Cluster Creation wizard. (Note that this includes clusters on AWS that were not created using the Cluster Creation wizard.) The screen capture below shows an overview of a 3-node on-premises cluster, which was created using the command line and imported into MC. This cluster has a running Enterprise Mode Vertica database on it.

Managing a Cluster Created on AWS with the Cluster Creation Wizard

Vertica Management Console with Provisioning, a version of MC launched using AWS resources, offers other cluster management operations that are specific to the cloud. In this way, you can manage a cluster running on AWS without going to the AWS console.

In the screen capture below, the Cluster page shows a 4-node cluster, which was provisioned using Vertica Management Console with Provisioning.
You can perform the following operations on your cluster using MC with Provisioning:

- **Start Cluster.** Start all the instances in the cluster. Available at the top of the Cluster page.

- **Stop Cluster.** Stop all the instances in the cluster. You must first stop any running database on the cluster. Available at the top of the cluster page.

- **Reboot Cluster.** Restart all instances in the cluster. Available under the Advanced menu at the top of the page.

- **Terminate Cluster.** Terminate all instances in the cluster, the databases on it, and all AWS resources from the cluster. Note that this operation permanently deletes any data you had on the cluster or its databases. Available under the Advanced menu at the top of the page.

### Managing AWS Instances

You can also perform management operations on individual instances in your cluster. To do so, in the Instance List panel, select the IP address of any instance in your cluster that you want to
perform the action on. Then click an icon from the menu at the top of the panel. Hover over an icon to read the action it performs.

- **Start Instance.** Start an individual instance in the cluster.
- **Stop Instance.** Stop an individual instance in the cluster.
- **Add Instance.** Add another instance to your cluster. When you select this action, MC opens the Add AWS Instance wizard, where you will specify volume and storage information for the instance. You must supply your AWS key pair (and a Vertica Premium Edition license if you are adding more nodes to the cluster than a Community Edition license allows).
- **Restart Instance.** Restart an individual instance in the cluster.
- **Terminate Instance.** Permanently remove the instance from your cluster.

---

Create an Empty Database Using MC

You can create a new database on an existing Vertica cluster through the Management Console interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign back in to MC later; the creation process continues unless an unexpected error occurs. See the Notes section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in Configuration Procedure. You should also run the Database Designer, which you access through the Administration Tools, to create either a comprehensive or incremental design. Consider using the Tutorial in Getting Started to create a sample database you can start monitoring immediately.
How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the Databases and Clusters page, skip to the next step; otherwise:
   a. **Connect** to MC and sign in as an MC administrator.
   b. On the **Home page**, click Existing Infrastructure to view the Databases and Clusters page.

2. If no databases exist on the cluster, continue to the next step; otherwise:
   a. If a database is running on the cluster on which you want to add a new database, select the database and click Stop.
   b. Wait for the running database to have a status of **Stopped**.

3. Click the cluster on which you want to create the new database and click Create Database.

4. The Create Database wizard opens. Provide the following information:
   - Database name and password. See [Creating a Database Name and Password](#) for rules.
   - Optionally click Advanced to open the advanced settings and change the port, catalog path, and data path. By default the MC application/web server port is 5450 and paths are `/home/dbadmin`, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the `install_vertica` script. Do not use the default agent port 5444 as a new setting for the MC port. See [MC Settings > Configuration for port values](#).

5. Click Continue.

6. Select nodes to include in the database.

   The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node and clear Include host in new database to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the Include check box.

7. Click Create in the Database Configuration window to create the database on the nodes.
The creation process takes a few moments, after which the database starts and a Success message appears on the interface.

8. Click OK to close the success message.

The Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your Vertica cluster, which are not shown in standby mode, you must run the install_vertica script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
  - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database Message Center from the MC Home page.

  - Failure messages display on the database Manage page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.

  - To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the Node Details page.

- To create MC users and grant them access to an MC-managed database, see About MC Users and Creating an MC User.

See Also

- Creating a Cluster Using MC
- Troubleshooting with MC Diagnostics
- Restarting MC
Importing an Existing Database Into MC

If you have already created a Vertica database, you can import it into MC to monitor its health and activity.

When you install MC on the same cluster as the existing database you intend to monitor, MC automatically discovers the cluster and any databases installed on it, whether those databases are currently running or are down.

Note: If you haven't created a database and want to create one through the MC, see Create an Empty Database Using MC.

Import a Database Existing on a Monitored Cluster

The following procedure describes how to import an existing database that is on a cluster MC is already monitoring.

1. Connect to Management Console and sign in as an MC administrator.

2. On the MC Home page, click View Your Infrastructure.

3. On the Databases and Clusters page, click the cluster and click View in the dialog box that opens.

4. On the left side of the page under the Databases heading, and click Import Discovered.

   Tip: A running database appears as Monitored; Sany non-running databases appear as Discovered. MC supports only one running database on a single cluster at a time. You must shut down a running database on a cluster in order to monitor another database on that cluster.

5. In the Import Database dialog box:
   a. Select the database you want to import.
   b. Optionally clear auto-discovered databases you don't want to import.
c. Supply the database administrator username and password and click Import.
   (Supplying a non-administrator username prevents MC from displaying some charts after import.)

After Management Console connects to the database it opens the Manage page, which provides a view of the cluster nodes. See Monitoring Cluster Status for more information.

You perform the import process once per existing database. Next time you connect to Management Console, your database appears under the Recent Databases section on the Home page, as well as on the Databases and Clusters page.

**Note:** The system clocks in your cluster must be synchronized with the system that is running Management Console to allow automatic discovery of local clusters.

## Import a Database Existing on a New Cluster

If the database you intend to monitor is on a cluster MC is not currently monitoring, MC cannot automatically discover it. You can import it with the following procedure.

1. **Connect** to Management Console and sign in as an MC administrator.

2. On the MC **Home page**, click **Import a Vertica Database Cluster**.

3. Enter the IP address of one of the database's cluster nodes.

4. Enter the master API key for the cluster. Find the key here: `/opt/vertica/config/apikeys.dat`

5. In the **Import Database** dialog box:
   a. Select the database you want to import.
   b. Optionally clear auto-discovered databases you don't want to import.
   c. Supply the database administrator username and password and click Import.
      (Supplying a non-administrator username prevents MC from displaying some charts after import.)
Import and Monitor in a Hadoop Environment

You can use Management Console to connect to and monitor a Vertica database that resides in an Apache Hadoop environment.

Connecting to Ambari and Importing Vertica Within a Hadoop Environment

To import your Vertica database that resides in a Hadoop environment, connect to that Hadoop environment in Management Console through an Apache Ambari server. Then, through that connection, import Vertica.

Before you connect and import, you must:

- Install Vertica on a Hadoop cluster
- Install Apache Ambari version 1.6.1 or 2.1.0
- Enable Ganglia on your Hadoop cluster, to get the most information from your Hadoop environment

To connect to an Ambari server:

1. From the Management Console home page, select Provisioning. Then, click Connect using an Ambari server to import Vertica within a Hadoop environment.
2. In the Provision Access within Hadoop Environment window, enter a new Ambari connection with your username and password. If you have already have an existing connection, a window opens that allows you to select an existing connection from the drop-down menu.

3. In the next window, select the Hadoop cluster where the Vertica database you want to monitor resides. MC will discover Hadoop clusters that are currently monitored by the Ambari server you specify.

   A window appears confirming that your Hadoop cluster is saved. If MC currently does not monitor Vertica clusters in the specified Hadoop environment, you have the option to import Vertica clusters at this time.

4. Enter the IP address for the Vertica database you want to import and monitor. If Vertica is running on multiple hosts, enter the IP address of one of them.

5. Enter the API key for the Vertica cluster. The API key is generated during Vertica installation and you can find it in the /opt/vertica/config/apikeys.dat file.

6. When the next window displays the discovered databases, select one or more database you want to import, along with the database username and password.
7. Confirm that the import is successful. If it is, a success message appears. Click Done to go to the Existing Infrastructure page.

To import an additional Vertica cluster within a Hadoop environment, click Import Cluster or database using IP address discovery under Provisioning. Management Console will automatically associate the cluster with the existing Hadoop environment.

Monitoring Vertica Within a Hadoop Environment

To monitor the Vertica clusters in a Hadoop environment, navigate to the Existing Infrastructure page:
Click to select the Hadoop environment, and then click View Vertica Databases.

The resulting screen displays information about the Vertica databases that reside in a Hadoop environment:
You can monitor information like resource usage, Hadoop services, and database and connection status.

### Updating and Removing an Ambari Connection

To update or remove an existing Ambari connection, go to the MC Existing Infrastructure page, and click on the relevant Hadoop environment.

To update a connection, click **Update Ambari Connection**. Step through the wizard to update the connection.

To remove a connection, select **Update Ambari Connection** and choose **Remove Connection**, or click **Discontinue Monitoring** and then confirm that you want to remove the connection. Removing the connection also removes all Vertica databases associated with this connection from MC monitoring. You can re-import the databases later if needed.

### See Also:

- Integrating with Apache Hadoop
Managing MC Settings

The MC Settings page allows you to configure properties specific to Management Console. You can:

- Change MC and agent default port assignments.

- Enable and disable username and password auto-complete at MC login. (After disabling, clear your browser's cache.)

- Control the following monitoring settings in MC:
  - Enable checks and set alert thresholds for spread retransmit rate. This setting is disabled by default. The recommended alert threshold for spread retransmit rate is 10%.
  - Set alert thresholds for free MC disk space checks. The recommended alert threshold is 500 MB.
  - Exclude MC queries from activity charts.
  - Set refresh intervals for MC charts and pages.

- Upload a new SSL certificate.

- Use LDAP for user authentication.

- Create new MC users and, with their user credentials, map them to an database managed by MC on the Vertica server. See Creating an MC User and Managing MC Users.

- View your version of Vertica or upload a new Vertica binary file.

- Customize the look and feel of MC with themes. See Customizing Look and Feel.

- Configure MC to use an alternative data source to monitor your database. See Monitoring External Data Sources in Management Console.

- Enable MC to send email alerts. See Set Up Email.

- Configure Extended Monitoring, which allows you to monitor more long-term data in MC:
  - Set up an external storage database for Extended Monitoring. See Managing the Storage Database.
  - Enable or disable Extended Monitoring on your databases. See Managing Extended Monitoring on a Database.
Modifying Database-Specific Settings

To inspect or modify settings related to a database managed by MC, go to the Existing Infrastructure page. On this page, select a running database to see its Overview page. From the bottom of the Overview page, click the Settings tab to make modifications to database-specific settings.

Customizing Look and Feel

Management Console themes provide a unique look and feel to the Management Console interface. Access these themes through the Theme page in MC Settings.

Changing Themes

Only administrators, who have access to MC Settings, can alter themes. The selected theme is visible to all Management Console users.

To apply a new theme on Management Console, go to MC Settings > Theme. On the Theme page, select a theme from the drop-down menu. A preview of the selected theme appears. Click Apply in the upper right hand of the page to apply the change.
Available Themes

The OpenText Theme is the default Management Console theme. This theme includes the colors and styles used by [[Undefined variable _Branding_Variables._Company_Acronym]] branding.
The Vertica Classic theme features a dark background and green accents, evoking classic computer terminal interfaces.
Changing MC or Agent Ports

When you configure MC, the Configuration Wizard sets up the following default ports:

- 5450—Used to connect a web browser session to MC and allows communication from Vertica cluster nodes to the MC application/web server
- 5444—Provides MC-to-node and node-to-node (agent) communications for database create/import and monitoring activities

If You Need to Change the MC Default Ports

A scenario might arise where you need to change the default port assignments for MC or its agents. For example, perhaps one of the default ports is not available on your Vertica cluster, or you encounter connection problems between MC and the agents. The following topics describe how to change port assignments for MC or its agents.

See Also

- Ensure Ports Are Available

How to Change the Agent Port

Changing the agent port takes place in two steps: at the command line, where you modify the config.py file and through a browser, where you modify MC settings.

Change the Agent Port in config.py

1. Log in as root on any cluster node and change to the agent directory:

   ```
   # cd /opt/vertica/agent
   ```

2. Use any text editor to open config.py.
3. Scroll down to the `agent_port = 5444` entry and replace 5444 with a different port number.

4. Save and close the file.

5. Copy `config.py` to the `/opt/vertica/agent` directory on all nodes in the cluster.

6. Restart the agent process by running the following command:

   ```
   # /etc/init.d/vertica_agent restart
   ```

   **Note:** If you are using Red Hat Enterprise Linux/CentOS 7, use the following command instead:

   ```
   # /opt/vertica/sbin/vertica_agent restart
   ```

7. Repeat (as root) Step 6 on each cluster node where you copied the `config.py` file.

### Change the Agent Port on MC

1. Open a web browser and connect to MC as a user with MC ADMIN privileges.

2. Navigate to MC Settings > Configuration.

3. Change Default Vertica agent port from 5444 to the new value you specified in the `config.py` file.

4. Click Apply and click Done.

5. Restart MC so MC can connect to the agent at its new port. See Restarting MC.

### How to Change the MC Port

Use this procedure to change the default port for MC's application server from 5450 to a different value.
1. Open a web browser and connect to MC as a user with MC ADMIN privileges.

2. On the MC Home page, navigate to MC Settings > Configuration and change the Application server running port value from 5450 to a new value.

3. In the change-port dialog, click OK.

4. Restart MC.

5. Reconnect your browser session using the new port. For example, if you changed the port from 5450 to 5555, use one of the following formats:

   - https://00.00.00:5555/
   - https://hostname:5555/
**Backing Up MC**

Before you **upgrade MC**, Vertica recommends that you back up your MC metadata (configuration and user settings). Use a storage location external to the server on which you installed MC.

1. On the target server (where you want to store MC metadata), log in as root or a user with sudo privileges.

2. Create a backup directory as in following example:

   ```bash
   # mkdir /backups/mc/mc-backup-20130425
   ```

3. Copy the `/opt/vconsole` directory to the new backup folder:

   ```bash
   # cp –r /opt/vconsole /backups/mc/mc-backup-20130425
   ```
Troubleshooting with MC Diagnostics

The Management Console Diagnostics page, which you access from the Home page, helps you resolve issues within the MC process, not the database.

What You Can Diagnose

- View Management Console logs, which you can sort by column headings, such as type, component, or message).

  - Search within messages for key words or phrases and search for log entries within a specific time frame.

- Export database messages to a file.

- Reset console parameters to their original configuration.

  Caution: Reset removes all data (monitoring and configuration information) from storage and forces you to reconfigure MC as if it were the first time.

- Restart the Management Console process. When the process completes, you are directed back to the login page.

Viewing the MC Log

If you want to browse MC logs (not database logs), navigate to the Diagnostics > MC Log page.

This page provides a tabular view of the contents at /opt/vconsole/log/mc/mconsole.log, letting you more easily identify and troubleshoot issues related to MC.

You can sort log entries by clicking the column header and search within messages for key words, phrases, and log entries within a specific time frame. You can also export log messages to a file.
Exporting the User Audit Log

When an MC user makes changes on Management Console, whether to an MC-managed database or to the MC itself, their action generates a log entry that contains data you can export to a file.

If you perform an MC factory reset (restore MC to its pre-configured state), you automatically have the opportunity to export audit records before the reset occurs.

To Manually Export MC User Activity

1. From the MC Home page, click Diagnostics and then click Audit Log.

2. On the Audit log viewer page, click Export and save the file to a location on the server.

To see what types of user operations the audit logger records, see Monitoring MC User Activity Using Audit Log.
Restarting MC

You might need to restart the MC web/application server for a number of reasons, such as after you change port assignments, use the MC interface to import a new SSL certificate, or if the MC interface or Vertica-related tasks become unresponsive.

Restarting MC requires ADMIN Role (mc) or SUPER Role (mc) privileges.

How to Restart MC through the MC Interface (Using Your Browser)

1. Open a web browser and connect to MC as an administrator.
2. On MC's Home page, click Diagnostics.
3. Click Restart Console and then click OK to continue or Cancel to return to the Diagnostics page.

The MC process shuts down for a few seconds and automatically restarts. After the process completes, you are directed back to the sign-in page.

How to Restart MC at the Command Line

If you are unable to connect to MC through a web browser for any reason, such as if the MC interface or Vertica-related tasks become unresponsive, you can run the vertica-consoled script with start, stop, or restart arguments.

Follow these steps to start, stop, or restart MC.

1. As root, open a terminal window on the server on which MC is installed.
2. Run the `vertica-consoled` script:

   ```bash
   # /etc/init.d/vertica-consoled { stop | start | restart }
   
   For versions CentOS 7 and above, run:
   
   # systemctl { stop | start | restart } vertica-consoled
   ```
**Important:** The `systemctl` function requires you to both start and stop services explicitly. If you kill or stop the `vertica-consoled` process without using `systemctl stop`, you cannot start the MC process again with the original `systemctl start` command. Instead, you must run `systemctl stop vertica-consoled before running systmctl start vertica-consoled`.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>stop</strong></td>
<td>Stops the MC application/web server.</td>
</tr>
<tr>
<td><strong>start</strong></td>
<td>Starts the MC application/web server. <strong>Caution:</strong> Use <code>start</code> only if you are certain MC is not already running. As a best practice, stop MC before you issue the <code>start</code> command.</td>
</tr>
<tr>
<td><strong>restart</strong></td>
<td>Restarts the MC application/web server. This process will report that the stop didn't work if MC is not already running.</td>
</tr>
</tbody>
</table>

### How to Restart MC on an AMI

You can use the following steps to restart an MC AMI instance.

1. SSH into the MC host as user `dbadmin`:

   ```
   $ ssh -i example.pem dbadmin@52.xx.xx.xx
   ```

2. Run the `vertica-consoled` script using `sudo`:

   ```
   # sudo /etc/init.d/vertica-consoled { stop | start | restart }
   ```

### Starting over

If you need to return MC to its original state (a "factory reset"), see [Resetting MC to Pre-Configured State](#).
Resetting MC to Pre-Configured State

If you decide to reset MC to its original, preconfigured state, you can do so on the Diagnostics page by clicking Factory Reset.

Tip: Consider trying one of the options described in Restarting MC first.

A factory reset removes all metadata (about a week's worth of database monitoring/configuration information and MC users) from storage and forces you to reconfigure MC again, as described in Configuring MC in Installing Vertica.

After you click Factory Reset, you have the chance to export audit records to a file by clicking Yes. If you click No (do not export audit records), the process begins. There is no undo.

Keep the following in mind concerning user accounts and the MC.

- When you first configure MC, during the configuration process you create an MC superuser (a Linux account). Issuing a Factory Reset on the MC does not create a new MC superuser, nor does it delete the existing MC superuser. When initializing after a Factory Reset, you must logon using the original MC superuser account.

- Note that, once MC is configured, you can add users that are specific to MC. Users created through the MC interface are MC specific. When you subsequently change a password through the MC, you only change the password for the specific MC user. Passwords external to MC (i.e., system Linux users and Vertica database passwords) remain unchanged.

  For information on MC users, refer to the sections, Creating an MC User and MC configuration privileges.

Avoiding MC Self-Signed Certificate Expiration

When you connect to MC through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See Set Up Time Synchronization in Installing Vertica.
Running Queries in Management Console

You can use the Query Runner to run SQL queries on your database through Management Console (MC). After executing a query, you can also get the query plan and profile information for the query on this page.

To reach the Query Runner, select your database from the Home page or the Databases and Clusters page to view your database's Overview page. Select Query Execution at the bottom of the Overview page.

To familiarize yourself with how queries work in Vertica, you can refer to the Queries section of the documentation, as well as the SQL Reference Manual.

Limitations

You cannot execute COPY LOCAL statements using the Query Runner. To do so, use the vsq1 client installed on the server. See Using vsq1. (To use MC to import data from Amazon S3
storage to your Vertica database, see Loading Data Using MC.)

Manually commit any transactions (INSERT and COPY statements) you perform by adding the COMMIT statement in the text box after the transaction statements. If you do not do so, the transaction rolls back.

In the following example, to insert values into table1, include a COMMIT statement in the text box and execute the two statements together:

```
INSERT INTO table1 VALUES (1,2);
COMMIT;
```

**Format**

To input a series of queries, delimit them with a semicolon (;).

To automatically format the SQL text you have input, click the Format icon (</>).

**Privileges**

It is important when running queries in MC that the database administrator has correctly set up MC user privileges. The administrator must map all MC user profiles to their corresponding database user.

The Query Runner only permits MC users to perform actions that their corresponding Vertica database roles allow.

To set up user mappings, go to Home > MC Settings > User Management.

For more about how mapping MC user profiles to database users works, see Granting Database Access to MC Users. For information about database-level users and privileges, see the Managing Users and Privileges section of the documentation.
Execute a Query

The Query Runner provides several ways to input a query to run:

- **Input text.** Enter the text for a query or series of queries into the text box.

- **Import a SQL script.** Click the Upload icon ( ) to the top right of the text box to upload a SQL script (plain text file, typically with an extension of .sql). The queries from that file appear in the text box.

- **Enter a previous query from the Query History tab.** The Query History tab, on the left side of the page, displays the last 100 queries you have executed using the Query Runner on your current device and browser. Click any previous query in this tab to enter that query into the text box.

  Hover over a query in the Query History tab to view all the query text. To clear queries from your history, hover over an individual query and click x, or click Clear all at the top of the tab. Click the star to the left of any query to favorite it, so it won’t be cleared when you click Clear all.

Cick **Execute Query** to run the queries you have input.

You can also execute only a portion of the text entered into the text box, as long as the selected text is a valid query. To do so, select that portion of the text. The Execute selected text as query button then appears below the text box.

For example, you might execute only a part of the entered text if you have uploaded a SQL script that containing multiple queries, but you decide to run only one of those queries.

To customize your execution settings, click the Settings icon ( ) at the top right of the text box:

- **Row Limit:** Set the maximum number of rows to return. By default, the limit is 10000 rows.

- **Search Path:** Specify the schema to query.
Get Query Results

The Query Runner returns results in a table format. If you ran multiple queries simultaneously, the results window displays a tab for each set of results. View the number of rows returned and the query execution time at the bottom of the results window.

If your result returns many columns, you can click Auto-resize all columns in the top right of the results window for a better fit, or click and drag column borders to manually resize individual columns.

Sort results by clicking on a column name, or use the search bar to narrow down results.

Query Plans and Profiles

Each query result also displays an option to retrieve the plan or profile for that query.
After retrieving a plan or profile, you can expand or collapse the results view to see different levels of detail. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata, if it is available.

Note that the Query Runner does not automatically provide query profiles for queries that run for less than 1 second. To do so, prepend the word PROFILE to the query and run it.

You can also profile your query on the Query Plan page. The Query Plan page provides more details about both plan and profile results, including a query plan drilldown by node, a tree path view, and a profile analysis.

### Keyboard Shortcuts

The Query Runner provides the following keyboard shortcuts:

- **?:** Press the question mark to display or dismiss a list of the available keyboard shortcuts. (You can also click the question mark icon at the top right of the text box to view this list.)

- **alt + ↑:** Press alt + up arrow to decrease the height of the text box.

- **alt + ↓:** Press alt + down arrow to increase the height of the text box.

- **ctrl + enter:** Press ctrl + enter to run the query.

- **ctrl + shift + enter:** Press ctrl + shift + enter to run selected text.
See Also

- Granting Database Access to MC Users
- Managing Users and Privileges
- Queries
- SQL Reference Manual
- Using vsq1
Managing Queries in MC

Management Console can show you a query plan in easy-to-read format, where you can review the optimizer's strategy for executing a specific query. You can view a query plan in either of two ways:

- View the plan of an active query.
- View the plan for any query that you manually specify.

Access the Plan of an Active Query

1. At the bottom of the Management Console window, click the Activity tab.
2. From the list at the top of the page, select Queries.
3. On the activity graph, click the data point that corresponds to the query you want to view.
4. In the View Plan column, click Explain next to the command for which you want to view the query plan. Only certain queries use query plans—for example, SELECT, INSERT, DELETE, and UPDATE.
5. In the Explain Plan window, click Explain. Vertica generates the query plan.
6. (Optional) View the output in Path Information view or Tree Path view. To do so, click the respective view buttons on the left of the output box.

Access the Plan for a Specific Query

1. Locate the query for which you want to see the query plan in either of the following ways:
   - Queries Not Running — In the Explain window, type or paste the query text into the text box.
   - Queries Currently Running — In the Find a Query By ID input window, perform one of the following actions:
○ Enter the query statement and transaction ID.

○ Click the Browse Running Queries link.

Caution: Entering the word EXPLAIN before the query results in a syntax error.

2. Click Explain. Vertica generates the plan.

   If the query is invalid, Management Console highlights in red the parts of your query that might have caused a syntax error.

3. (Optional) View the output in Path Information view or Tree Path view. To do so, click the respective view buttons on the left of the output box.

Accessing Query Plans in Management Console

You can access query plans in Management Console in two ways:

- In the Detail page for query-related charts on the database's Activity page, click Explain next to a query to view a plan for that query.

- Enter a query manually on the Explain page and click Explain Plan.

In both cases, the following window opens:
You can also enter the transaction ID and statement ID or browse running or completed queries in the Find a Query input window:

In the output window, you can perform the following tasks related to the query you entered:

- Expand and collapse query paths.
- Clearing query data.
- View projection and column metadata.
- Use different query plan views.
Query Plan View Options

Vertica Management Console provides two views for displaying query plans:

- Path Information
- Tree Path

Note: Query Plan Drilldown and Profile Analysis output views are only available when you run PROFILE.

You can change the query plan view using the icons on the bottom portion of the Explain page.

The Path Information view displays the query plan path. You can expand or collapse the view to see different levels of detail. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata, if it is available.
The Tree Path view details the query plan in the form of a tree. When you run EXPLAIN, the tree view does not contain any metrics because the query has not yet executed.

Expanding and Collapsing Query Paths

The EXPLAIN window initially displays the full query plan as generated by the EXPLAIN command. Query plans can be lengthy, so you might want to modify the display so you can focus only on areas of interest:

- **Collapse All** collapses all query paths, and displays only a summary of each path.
- **Expand All** expands all query paths.
- **Click** the first line of a path to display details for that path. To collapse that path, click its first line again.

For details about EXPLAIN command output, see EXPLAIN-Generated Query Plans.

Clearing Query Data

After you finish reviewing the current query data, click Clear All to clear the query text and data. Alternatively, to display information about another query, enter the query text and click
Explain or Profile.

Viewing Projection and Column Metadata

In the Management Console EXPLAIN window, when query paths are expanded in the Path Information view, Projection lines contain a projection name and Materialize lines contain one or more column names.

To view metadata for a projection or a column, click the object name. A pop-up window displays the metadata. The following image on the left shows example projection metadata and the image on the right shows example column metadata.

**Note:** Most system tables do not have metadata.

When you are done viewing the metadata, close the pop-up window.
Creating a Database Design in Management Console

Database Designer creates an design that provides excellent performance for ad-hoc queries and specific queries while using disk space efficiently. Database Designer analyzes the logical schema definition, sample data, and sample queries, and creates a physical schema that you can deploy.

For more about how Database Designer works, see the Creating a Database Design section of the documentation, and About Database Designer.

To use Management Console to create an optimized design for your database, you must be a DBADMIN user or have been assigned the DBDUSER role.

Management Console provides two ways to create a design:

- Wizard—This option walks you through the process of configuring a new design. Click Back and Next to navigate through the Wizard steps, or Cancel to cancel creating a new design.

  To learn how to use the Wizard to create a design, see Using the Wizard to Create a Design.

- Manual—This option creates and saves a design with the default parameters.

  To learn how to create a design manually, see Creating a Design Manually.

Tip: If you have many design tables that you want Database Designer to consider, it might be easier to use the Wizard to create your design. In the Wizard, you can submit all the tables in a schema at once; creating a design manually requires that you submit the design tables one at a time.

Using the Wizard to Create a Design

Take these steps to create a design using the Management Console's Wizard:

1. On your database's dashboard, click the Design tab at the bottom of the page to navigate to the Database Designer page.
The left side of the Database Designer page lists the database designs you own, with the most recent design you worked on highlighted. That pane also lists the current status of the design. Details about the most recent design appear in the main pane.

The main pane contains details about the selected design.

2. To create a new design, click New Design.

3. Enter a name for your design, and click Wizard.

   For more information, see Design Name.

4. Navigate through the Wizard using the Back and Next buttons.
5. To build the design immediately after exiting the Wizard, on the Execution Options window, select Auto-build.

Important: Vertica does not recommend that you auto-deploy the design from the Wizard. There may be a delay in adding the queries to the design, so if the design is deployed but the queries have not yet loaded, deployment may fail. If this happens, reset the design, check the Queries tab to make sure the queries have been loaded, and deploy the design.

6. When you have entered all the information, the Wizard displays a summary of your choices. Click Submit Design to build your design.

See Also

- About Database Designer
- Creating a Design Manually
Creating a Design Manually

To create a design using Management Console and specify the configuration, take these steps.

1. On your database's dashboard, click the Design tab at the bottom of the page to navigate to the Database Designer page.

   The left side of the Database Designer page lists the database designs you own, with the most recent design you worked on highlighted. That pane also lists the current status of the design. Details about the most recent design appear in the main pane.

   The main pane contains details about the selected design.
2. To create a new design, click **New Design**.

3. Enter a name for your design and select **Manual**.

   The main **Database Design** window opens, displaying the default design parameters. Vertica has created and saved a design with the name you specified, and assigned it the default parameters.

   For more information, see **Design Name**.

4. On the **General** window, modify the design type, optimization objectives, K-safety, **Analyze Correlations Mode**, and the setting that allows **Database Designer** to create unsegmented projections.
If you choose Incremental, the design automatically optimizes for the desired queries, and the K-safety defaults to the value of the cluster K-safety; you cannot change these values for an incremental design.

Analyze Correlations Mode determines if Database Designer analyzes and considers column correlations when creating the design. For more information, see `DESIGNER_SET_ANALYZE_CORRELATIONS_MODE`.

5. Click the Tables tab. You must submit tables to your design.

6. To add tables of sample data to your design, click Add Tables. A list of available tables appears; select the tables you want and click Save. If you want to remove tables from your design, click the tables you want to remove, and click Remove Selected.

   If a design table has been dropped from the database, a red circle with a white exclamation point appears next to the table name. Before you can build or deploy the design, you must remove any dropped tables from the design. To do this, select the dropped tables and and click Remove Selected. You cannot build or deploy a design if any of the design tables have been dropped.

7. Click the Queries tab. To add queries to your design, do one of the following:
   - To add queries from the `QUERY_REQUESTS` system table, click Query Repository. In the Queries Repository dialog that appears, you can sort queries by most recent, most frequently executed, and longest running. Select the desired queries and click Save. All valid queries that you selected appear in the Queries window.
   - To add queries from a file, select Upload. All valid queries in the file that you select are added to the design and appear in the Queries window. (This option is only available for the MC super administrator role.)

   Database Designer checks the validity of the queries when you add the queries to the design and again when you build the design. If it finds invalid queries, it ignores them.

   If you have a large number of queries, it may take time to load them. Make sure that all the queries you want Database Designer to consider when creating the design are listed in the Queries window.

8. Once you have specified all the parameters for your design, you should build the design. To do this, select your design and click Build Design.
9. Select Analyze Statistics if you want Database Designer to analyze the statistics before building the design.

   For more information see Statistics Analysis.

10. If you do not need to review the design before deploying it, select Deploy Immediately. Otherwise, leave that option unselected.

11. Click Start. On the left-hand pane, the status of your design displays as Building until it is complete.

12. To follow the progress of a build, click Event History. Status messages appear in this window and you can see the current phase of the build operation. The information in the Event History tab contains data from the OUTPUT_EVENT_HISTORY system table.

13. When the build completes, the left-hand pane displays Built. To view the deployment script, select your design and click Output.

14. After you deploy the design using Management Console, the deployment script is deleted. To keep a permanent copy of the deployment script, copy and paste the SQL commands from the Output window to a file.

15. Once you have reviewed your design and are ready to deploy it, select the design and click Deploy Design.

16. To follow the progress of the deployment, click Event History. Status messages appear in this window and you can see the current phase of the deployment operation.

   In the Event History window, while the design is running, you can do one of the following:

   ■ Click the blue button next to the design name to refresh the event history listing.
   ■ Click Cancel Design Run to cancel the design in progress.
   ■ Click Force Delete Design to cancel and delete the design in progress.

17. When the deployment completes, the left-hand pane displays Deployment Completed. To view the deployment script, select your design and click Output.

   Your database is now optimized according to the parameters you set.
Getting Tuning Recommendations in MC

If queries perform sub-optimally, you can get tuning recommendations for them, as well as for hints about optimizing database objects, by using the Workload Analyzer (WLA).

WLA is a Vertica utility that analyzes system information in Vertica system tables. It then returns a set of tuning recommendations based on statistics, system and data collector events, and database/table/projection design. You can use these recommendations to tune query performance.

View WLA Recommendations

WLA recommendations are available through the database's Overview page.

1. On the right side of the Overview page in the Quick Stats bar, view the Workload Analyzer module. This module alerts you to the number of recommendations WLA has generated.

2. Click the number to view the WLA recommendations on the Database Designer page. The Workload Analyzer Results dialog displays tuning recommendations and the resource cost of running each command (Low, Medium, or High).

For additional information about tuning recommendations and cost see Analyzing Workloads in the Administrator's Guide and ANALYZE_WORKLOAD in the SQL Reference Manual.

The following image shows a list of WLA tuning recommendations for a database. In the output, WLA advises that you run the Database Designer on several tables and provides the cost estimate of running these operations.

Note that the cost of running Database Designer is high. When you see a high cost, you might want to run the recommended tuning action after hours.
MC displays fifteen recommendations per page and includes a message at bottom left, which lets you know where you are within the total number of recommendations; for example, "Showing n-nn of nn entries." Depending on the total number of recommendations, you can move forward and backward through them by clicking the arrows at bottom right.

Tip: You can force the WLA task to run immediately by clicking Update Recommendations over the Cost column.

Set and Disable WLA Events

On MC, WLA automatically begins monitoring data one minute after the MC process starts. WLA then runs once per day, or immediately after you import a database to MC. It continually gathers data in the background as long as the database is running. If you haven't created a database yet, or if the database is down, WLA does nothing until the database is back up.
By default, WLA runs each day at 2 AM. To optimize when WLA uses resources, you can set WLA to run at a different time for any or all databases that MC monitors. You can also set MC to never run WLA automatically.

1. On the Home page, click **MC Settings**.

2. Click the **Monitoring** tab.

3. Under the **Workload Analyzer Assistant** section of the Monitoring page, select your timezone.

4. Select the radio button for one of the options:
   - **All Databases**: Select a time from the list. WLA will run at that time on all databases that MC monitors.
   - **Specific Database at Specific Time**: Select a database and a time from the list. At the time you specify, WLA will run at that time on the database you selected.
   - **Do Not Run Workload Analyzer On Any Database**: MC will never run WLA automatically on any database it monitors.

5. Click **Apply** at the top right of the page.

For additional information about tuning recommendations and their triggering event, see [Understanding WLA Triggering Conditions](#).

### See Also

- [Analyzing Workloads](#)
- [Getting Tuning Recommendations](#)
Monitoring Using MC

Management Console gathers and retains history of important system activities about your MC-managed database cluster, such as performance and resource utilization. You can use MC charts to locate performance bottlenecks on a particular node, to identify potential improvements to Vertica configuration, and as a reference for what actions users have taken on the MC interface.

Note: MC directly queries Data Collector tables on the MC-monitored databases themselves. See Management Console Architecture in Vertica Concepts. For how to set up MC to query an alternative database for monitoring data, see Extended Monitoring.

The following list describes some of the areas you can monitor and troubleshoot through the MC interface:

- Multiple database cluster states and key performance indicators that report on the cluster's overall health
- Information on individual cluster nodes specific to resources
- Database activity in relation to CPU/memory, networking, and disk I/O usage
- Query concurrency and internal/user sessions that report on important events in time
- Cluster-wide messages
- Database and agent log entries
- MC user activity (what users are doing while logged in to MC)
- Issues related to the MC process
- Error handling and feedback

About Chart Updates

MC charts update dynamically with text, color, and messages Management Console receives from the agents on the database cluster. This information can help you quickly resolve problems.

Each client session to MC uses a connection from MaxClientSessions, a database configuration parameter. This parameter determines the maximum number of sessions that
can run on a single database cluster node. Sometimes multiple MC users, mapped to the same database account, are concurrently monitoring the Overview and Activity pages. In such cases, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to account for extra sessions. See Managing Sessions for details.

**Monitoring Same-Name Databases on MC**

If you are monitoring two databases with identical names on different clusters, you can determine which database is associated with which cluster by clicking the database icon on MC's Databases and Clusters page to view its dialog box. Information in the dialog displays the cluster on which the selected database is associated.
Viewing the Overview Page

The Overview page displays a dynamic dashboard view of your database.

The page provides three tabs: Status Summary, System Health, and Query Synopsis. Each tab contains charts and filters displaying information about your cluster. The QuickStats widgets on the right of the page display alerts and statistics about the state of your cluster.

Information on this page updates every minute, but you can postpone updates by deselecting Auto Refresh Charts in the toolbar.

Chart Viewing Options

You can specify time frames for some charts, which display a calendar icon in their title bars. Click the calendar icon to specify the time frame for that module.

On the Status Summary tab, you can select Synchronize charts to simultaneously apply the specified time frame to all charts on that tab.

If you have enabled extended monitoring on your database, MC can display longer ranges of data in certain charts. See Extended Monitoring. If a chart is using extended monitoring data, the rocket ship icon appears in the title bar:

You can expand some charts to view them in larger windows. Click the expand icon in the title bar to do so:
Quick Stats

The Quick Stats sidebar on the right of the page provides instant alerts and information about your cluster's status.

| Database Nodes Health          | Total: 6  
|--------------------------------|-----------
| Down                          | 0         
| Critical                      | 0         
| Recovering                    | 0         
| Up                             | 6         
| Version: 9.0.0-0              |           

| Running And Queued Queries    | Total: 29  
|--------------------------------|-----------
| Running                       | 0         
| Queued                        | 0         

| Projections                   | Total: 18  
|--------------------------------|-----------
| Unsegmented                   | 0         
| Unsafe/Not Up                 | 0         
| For Largest Schema: public    |           

| Disk Space Usage              | Total: 6 Nodes  
|--------------------------------|-------------
| Based On Config Param For Alerts On Disk Usage: | >60%        
| Low                             |             

| Workload Analyzer             | Total: 33 Recommendations  
|--------------------------------|--------------------------

| I/O Wait Notices               | Total: 6 Nodes  
|--------------------------------|-------------
| Nodes with CPU wait time exceeding 1 sec | 0          
| Network nodes with network errors exceeding 0% | 0          
| Threshold value exceeded within last hour |           

| License Consumption            | Total: 1 Licenses  
|--------------------------------|-------------
| Vertical: 0.00% License        |             
| convaricaflexicore: N/A       |             

| Unread Messages (This Week)   | Total Unread Messages: 411  
|--------------------------------|--------------------------
| Unread High Priority          |             

- **Database Nodes Health** displays which nodes are down, critical, recovering, or up. Critical and recovering nodes are included in the total nodes considered "up" by the database. Click a node value to open the Manage page.

- **Running and Queued Queries** displays current queries in the database. Click the query values to open the Query Monitoring charts.

- **Projections** displays the number of total projections, unsegmented projections, and unsafe projections for the database schema with the most projections. Click a value to open the Table Treemap chart.

- **Disk Space Usage** alerts you to the number of nodes that are low on disk space. Click the value to go to the Manage page. On the Manage page, the Storage Used KPI View is displayed.

- **Workload Analyzer (WLA)** analyzes system information retained in SQL system tables and provides tuning recommendations, along with the cost (low, medium, or high) of running the command. See Analyzing Workloads for more information.

- **I/O Wait Notices** displays the number of nodes that, in the last hour, have recorded Disk I/O waits and Network I/O waits exceeding the wait threshold (1 second for Disk and 0 seconds for Network).

- **License Consumption** displays the number of licenses your database uses, and the percentage of your Vertica Community Edition or Premium Edition license being used.

- **Unread Messages** display the number of unread messages and alerts for the database. This count differs from the number of total messages across all your databases. Click the value to open the Message Center.
Status Summary

The Status Summary tab displays four modules that provide a general overview of the status of your cluster:

- The CPU/Memory/Disk I/O Usage module shows cluster resource usage. The chart displays the number of nodes in the database cluster and plots average and per-node percentages for CPU, memory, and disk I/O usage. Select a resource type from the legend to remove or add it from the chart display.

  Click a data point (which represents a node) to open the Manage page. See Monitoring Cluster CPU/Memory.

- The General Pool Activity module displays GENERAL pool activity. The chart displays average query queue times, average GENERAL pool free memory, and resource rejections. Use this chart to see how much free memory there is in GENERAL pool, or if there have been high queue times. Click the expand icon in the title bar to open the chart in a bigger window.

  Click a data point to open the Resource Pools Monitoring chart. See Managing Workloads.

- The Thresholds Notifications module displays alerts generated when a threshold has been exceeded in the database. Notifications are categorized by System Health and Performance.
In the module, you can acknowledge an alert (which marks it as read) or click the X to stop monitoring that threshold (which stops you receiving similar alerts in the future).

Customize thresholds and alert priorities for these notifications in the Thresholds tab of the database Settings page. See Customizing Threshold-Based Notifications.

- The Queries module displays query statistics. The first pie chart displays running and queued queries in the last 24 hours. The second chart displays completed and failed queries for the time frame you specify.

Click a query count number above the chart to open the Query Monitoring chart. See Monitoring Running Queries.

System Health

The System Health tab provides a summary of your system resource usage and node information, with filters that allow you to view resource usage within the ranges you specify.

**Note:** Adjusting the filters on the System Health tab does not affect any database or MC settings.

- The Memory Usage filter displays the number of nodes with high and low memory usage. Move the sliders to adjust the memory usage range filter.
For example, if you specify a range of 25% to 75% memory usage, the filter displays how many nodes are using less than 25% of memory (Low) and how many are using more than 75% (High). Hover your cursor over the Low and High values to see lists of what nodes fall, respectively, below or above the memory usage range you specified.

Click a node value to go to the Manage page, which displays the Memory Utilization KPI View.

- The Spread Retransmission Rate filter displays the number of nodes with high spread retransmission rates. When a node's retransmission rate is too high, it is not communicating properly with other nodes. Move the slider to adjust the retransmission rate filter.

Hover your cursor over the Nodes value to see a list of what nodes exceeded the spread retransmission rate you specified. Click the node value to view spread retransmit rate alerts in the Message Center.

- The CPU Usage chart displays the number of nodes with high and low CPU usage. Move the sliders to adjust the CPU usage range filter. Hover your cursor over the Low and High values to see lists of what nodes are below or above range you specified.

Click a node value to go to the Manage page, which displays the CPU Utilization KPI View.

- The Reboot Rate filter displays the number of times nodes in the cluster have rebooted within the specified time frame. Use this filter to discover if nodes have gone down recently, or if there have been an unusual number of reboots. Move the slider to adjust the number of days. Hover over the Times value to see a list of the nodes that have rebooted and the times at which they did so.

- The Disk Space Usage filter displays the number of nodes with high disk space usage. Move the slider to adjust the disk usage filter. Hover your cursor over the Nodes value to see a list of what nodes exceed the acceptable range.

Click the nodes value to go to the Manage page, which displays the Storage Used KPI View.

- The Cluster Clock Skew Rate module displays the number of nodes that exceed a clock skew threshold. Nodes in a cluster whose clocks are not in sync can interfere with time-related database functions, the accuracy of database queries, and Management Console's monitoring of cluster activity.
Query Synopsis

The Query Synopsis page provides two modules that report system query activity and resource pool usage:

- The Query Statistics module displays four bar charts that provide an overview of running, queued queries, failed, and completed queries in the past 24 hours. Select one of the options at the top of the module to group the queries by Resource Pools, Users, or Nodes.

  Click a bar on the chart to view details about those queries the Query Monitoring activity chart.

- The User Query Type Distribution chart provides an overview of user and system query activity. The chart reports the types of operation that ran. Hover your cursor over chart points for more details. Select a type of operation from the legend to remove or add it from the chart display. To zoom to a certain time frame, you can adjust the sliders at the bottom of the chart.

  Click a bar in the graph to open the Queries chart.
Monitoring Cluster Nodes

For a visual overview of all cluster nodes, click the running database on the Databases and Clusters page and click the Manage tab at the bottom of the page to open the cluster status page.

The cluster status page displays the nodes in your cluster.

![Cluster Node Status Preview](Vertica Management Console)

The appearance of the nodes indicate the following states:

- **Healthy**: The nodes appear green.
- **Up**: A small arrow to the right of the node points upward.
- **Critical**: The node appears yellow and displays a warning icon to the right.
- **Down**: The node appears red. To the right of the node, a red arrow points downwards.
- **Unplugged**: An orange outlet and plug icon appears to the right.

You can get information about a particular node by clicking it, an action that opens the node details page.
Filtering What You See

If you have a large cluster, where it might be difficult to view dozens to hundreds of nodes on the MC interface, you can filter what you see. The Zoom filter shows more or less detail on the cluster overall, and the Health Filter lets you view specific node activity; for example, you can slide the bar all the way to the right to show only nodes that are down. A message next to the health filter indicates how many nodes in the cluster are hidden from view.

On this page, you can perform the following actions on your database cluster:

- Add, remove and replace nodes
- Rebalance data across all nodes
- Stop or start (or restart) the database
- Refresh the view from information MC gathers from the production database
- View key performance indicators (KPI) on node state, CPU, memory, and storage utilization (see Monitoring Cluster Performance for details)

Note: Starting, stopping, adding, and dropping nodes and rebalancing data across nodes works with the same functionality and restrictions as those same tasks performed through the Administration Tools.

If You don't See What You Expect

If the cluster grid does not accurately reflect the current state of the database (for example if the MC interface shows a node in INITIALIZING state, but when you use the Administration Tools to View Database Cluster State, you see that all nodes are UP), click the Refresh button in the toolbar. Doing so forces MC to immediately synchronize with the agents and update MC with new data.

Don't press the F5 key, which redisplayes the page using data from MC and ignores data from the agent. It can take several seconds for MC to enable all database action buttons.
Monitoring Node Activity

If a node fails on an MC-managed cluster or you notice one node is using higher resources than other cluster nodes—which you might observe when monitoring the Overview page—open the Manage page and click the node you want to investigate.

The Node Details page opens and provides summary information for the node (state, name, total memory, and so on), as well as resources the selected node has been consuming for the last three hours, such as average CPU, memory, disk I/O percent usage, network consumption in kilobytes, and the percentage of disk storage the running queries have been using. You can also browse and export log-level data from AgentTools and Vertica log files. MC retains a maximum of 2000 log records.

For a more detailed view of node activity, use the mouse to drag-select around a problem area in one of the graphs, such as the large spike in network traffic in the above image. Then hover over the high data point for a summary.
Monitoring Cluster Performance

Key Performance Indicators (KPIs) are a type of performance measurement that let you quickly view the health of your database cluster through MC's Manage page. These metrics, which determine a node's color, make it easy for you to quickly identify problem nodes.

Metrics on the database are computed and averaged over the latest 30 seconds of activity and dynamically updated on the cluster grid.

How to Get Metrics on Your Cluster

To view metrics for a particular state, click the menu next to the KPI View label at the bottom of the Manage page, and select a state.

MC reports KPI scores for:
Node state—(default view) shows node status (up, down, k-safety critical) by color; you can filter which nodes appear on the page by sliding the health filter from left to right.

- CPU Utilization—average CPU utilization
- Memory Utilization—average percent RAM used
- Storage Utilization—average percent storage used

After you make a selection, there is a brief delay while MC transmits information back to the requesting client. You can also click Sync in the toolbar to force synchronization between MC and the client.

Node Colors and What They Mean

Nodes in the database cluster appear in color. Green is the most healthy and red is the least healthy, with varying color values in between.

Each node has an attached information dialog box that summarizes its score. It is the score's position within a range of 0 (healthiest) to 100 (least healthy) that determines the node's color bias. Color bias means that, depending on the value of the health score, the final color could be slightly biased; for example, a node with score 0 will be more green than than a node with a score of 32, which is still within the green range but influenced by the next base color, which is yellow. Similarly, a node with a score of 80 appears as a dull shade of red, because it is influenced by orange.

MC computes scores for each node's color bias as follows:

- 0-33: green and shades of green
- 34-66: yellow and shades of yellow
- 67-100: red and shades of red shades

If the unhealthy node were to consume additional resources, its color would change from a dull orange-red to a brighter red.

Filtering Nodes From the View

The health filter is the slider in the lower left area of page. You can slide it left to right to show or hide nodes; for example, you might want to hide nodes with a score smaller that a certain value so the UI displays only the unhealthy nodes that require immediate attention. Wherever
you land on the health filter, an informational message appears to the right of the filter, indicating how many nodes are hidden from view.

Filtering is useful if you have many nodes and want to see only the ones that need attention, so you can quickly resolve issues on them.

Monitoring Cluster CPU/Memory

On the MC Overview page, the CPU/Memory subsection provides a graph-based overview of cluster resources during the last hour, which lets you quickly monitor resource distribution across nodes.

This chart plots average and per-node percentages for both CPU and memory with updates every minute—unless you clear Auto Refresh Charts in the toolbar. You can also filter what the chart displays by clicking components in the legend at the bottom of the subsection to show/hide those components. Yellow data points represent individual nodes in the cluster at that point in time.

Investigating Areas of Concern

While viewing cluster resources, you might wonder why resources among nodes become skewed. To zoom in, use your mouse to drag around the problem area surrounding the time block of interest.

After you release the mouse, the chart refreshes to display a more detailed view of the selected area. If you hover your cursor over the node that looks like it's consuming the most resources, a dialog box summarizes that node's percent usage.

For more information, click a data point (node) on the graph to open MC's node details page. To return to the previous view, click Reset zoom.

See Also

- Monitoring Node Activity
Monitoring Database Storage

The Infrastructure page's Storage View provides a summary of the amount of data stored across your database, and where that data is persistently located. Use this view to monitor how much of your storage capacity your databases are using.

For a database running in Eon Mode Beta, MC displays bar charts that illustrate the subscription status of the database's shards and nodes, respectively. Use these charts to determine if your current subscription layout is optimal for querying your Eon Mode Beta database. For information about viewing subscription status charts, see Monitoring Subscription Status in Eon Mode Beta.

Monitor Storage Usage

The storage summary table lists all databases currently monitored by MC and information about their storage:

- **Database Size.** Click Load Size to calculate the total size of the database.
- **Database Mode.** Vertica databases run in Enterprise Mode, or Eon Mode Beta.
• **Storage Type.** Enterprise Mode databases list the OS of the local nodes where data is stored. Eon Mode Beta databases list the type of communal storage location where it stores its data. Eon Mode Beta currently only supports Amazon S3 locations.

• **Storage Locations.** Enterprise Mode databases list the local directory where data is persistently stored. Eon Mode Beta databases list the Amazon S3 bucket where it persistently stores its data.

• **Action.** Click Details. For Enterprise Mode databases, MC opens a dialog listing storage paths on each node. For Eon Mode Beta databases, MC displays charts at the bottom of the Storage View page, illustrating subscription statuses.

In front of Eon Mode Beta database names in the list, a plus icon displays. Click the icon to expand more details about the database's depot capacity and usage. The depot is cache-like storage where Eon Mode Beta databases keep local copies of communal storage data for faster query access. Click Percentage Used to view the Depot Activity chart for that database.

### Depot Information

When running in Eon Mode Beta the following information appears:

• **Read Depot.** Indicates whether or not the database read mode is enabled.

• **Write Depot.** Indicates whether or not the database write mode is enabled.

• **Total Depot Capacity Across Cluster.** The amount of storage space available for all nodes in the cluster.

• **Depot in Use Across Cluster.** The amount of storage space presently in use for all nodes in the cluster.

• **Percentage Used.** The percentage of storage space presently in use for all nodes in the cluster.

• **View Depot on Nodes.** Displays the Depot Location Details dialog that contains the following:
  - Node name.
  - Amount of storage in use on the depot.
  - Percentage of storage in use on the depot.
- The depot's storage limit.
- The path on the node where the depot resides.

## Monitoring Subscription Status in Eon Mode Beta

To view subscription charts for any Eon Mode Beta database you monitor, click View Your Infrastructure on the MC Home page. Then click the Storage View tab.

Click the Details action for that database in the storage summary list (highlighted in red in the image below).

When you click Details, two charts become available on the bottom half of the page: The Sharding Subscription chart, and the Node Subscription chart. You can switch between these two charts using the drop down menu to the right of the chart title.
Why Monitor Shard and Node Subscriptions?

Shards are segments of the data that is stored persistently in your Eon Mode Beta database's communal storage location on Amazon S3. Each node in the database subscribes to a subset of those shards. In this way, the node gets updated on when to populate its depot with new data from communal storage. (See Shards and Subscriptions.)

For K-safety in an Eon Mode Beta database, shards should have multiple node subscribers to ensure that even if a node goes down or is being used by another query, the data on that shard is still available on other nodes. If a shard has no node subscribers, that could indicate that data loss is occurring.

Subscriptions go through several transitions, which are illustrated by colors in the subscription charts:

- **Pending** (Yellow). The node is ready to subscribe to a certain shard. It cannot yet serve queries because it is not actively subscribed to the shard yet.

- **Passive** (Blue/Teal). The node could potentially serve queries for a shard it is passively subscribed to, but its depot contents for that shard may not yet be up to date, which could negatively impact query performance. The passively subscribed node is waiting for an active node subscriber of the shard to send it the most recent data.
Active (Green). The node is actively subscribed to the shard, can load new data from communal storage, and can serve queries for data in that shard. The actively subscribed node sends data from that shard to other subscribed nodes.

Removing (Dark Red/Maroon). The node is unsubscribing from the shard. It may have the most recent data from that shard, but that state is temporary until data from that shard is cleaned up.

Inactive (Red). The subscribed node is down. It can no longer serve queries for that shard.

Monitor Sharding Subscription

The Sharding Subscription chart displays how many nodes are subscribed to each shard in your database, and what type of subscription it is.

You can hover over any bar in the chart to see which nodes are subscribed to the shard. Click on a subscription type in the legend to show or hide it in the chart display.

The example below shows the shard subscription status for a running Eon Mode Beta database. The database has three nodes that are up, and one node (Node 4) that has been added to the cluster, but is down.

![Shard Subscription Chart](image)

You can hover over any bar in the chart to see which nodes are subscribed to the shard. In this example, nodes 1 and 3 have active subscriptions to the first shard (green); nodes 1 and 2 to the second shard; and nodes 2 and 3 to the third shard.

The active subscriptions are evenly spread across the shards. This is a k-safe Eon Mode Beta database.
Node 4 was subscribed to two shards; however, because it is down, its subscriptions to the shards are now inactive (red).

Monitor Node Subscriptions

Use this chart to view how many shards each node in your database is subscribed to, and the state of those subscriptions. The number of shards each node is subscribed to should be about the same to prevent overworking any given node.

Hover over any bar to see the shards it is subscribed to. The color of the bar indicates the state of each subscription. Click on a subscription type in the legend to show or hide it in the chart display.

The example below shows the same database from the Sharding Subscription example above. Nodes 1 through 3 are each actively subscribed to two shards (green). At least two nodes are subscribed to every shard in the database (which you can double check using the Sharding Subscription chart), ensuring that even if one of the nodes is down or being used in a query, another node is still actively subscribed and can access the data of that shard.

Since Node 4 is down, the chart shows that both its shard subscriptions are now inactive.
See Also

- Shards and Subscriptions
- Elasticity in Vertica in Eon Mode Beta
- Peer-to-Peer Metadata Transfer in Eon Mode Beta

Monitoring System Resources

MC's Activity page provides immediate visual insight into potential problem areas in your database's health by giving you graph-based views of query and user activity, hardware and memory impact, table and projection usage, system bottlenecks, and resource pool usage.

Select one of the following charts in the toolbar menu:

- Queries
- Internal Sessions
- User Sessions
- Memory Usage
- System Bottlenecks
- User Query Phases
- Monitoring Table Utilization and Projections
- Query Monitoring
- Resource Pool Monitoring
- Monitoring Catalog Memory

How up to date is the information?

System-level activity charts automatically update every five minutes, unless you clear Auto Refresh in the toolbar. Depending on your system, it could take several moments for the charts to display when you first access the page or change the kind of resource you want to view.

Chart Viewing Options

You can specify time frames for some charts, which display a calendar icon in their title bars. Click the calendar icon to specify the time frame for that module.

If you have enabled extended monitoring on your database, MC can display longer ranges of data in certain charts. See Extended Monitoring. If a chart is using extended monitoring data, the rocket ship icon appears in the title bar:
You can expand some charts to view them in larger windows. Click the expand icon in the title bar to do so:

Monitoring Query Activity

The Queries chart displays information about query concurrency and average resource usage for CPU/memory, network activity, and disk I/O percent based on maximum rated bandwidth.

Hover over a data point for more information about percent usage for each of the resource types.
If you click a data point, MC opens a details page for that point in time, summarizing the number of user queries and system queries. This page can help you identify long-running queries, along with the query type. You can sort table columns and export the report to a file.

**Monitoring Key Events**

On the main Queries page, MC reports when a key event occurred, such as a Workload Analyzer or rebalance operation, by posting a WLA (Workload Analyzer) and/or RBL (rebalance) label on the resource section of the chart.

**Filtering Chart Results**

The default query concurrency is over the last hour. The chart automatically refreshes every five minutes, unless you clear the Auto Refresh option in the toolbar. You can filter results for 1 hour, 1 day, or up to 1 week, along with corresponding average resource usage. You can also click different resources in the legend to show or hide those resources.

To return to the main Queries page, use the slider bar or click the 1h button.

**Viewing More Detail**

To zoom in for detail, click-drag the mouse around a section or use the sliding selector bar at the bottom of the chart. After the detailed area displays, hover your cursor over a data point to view the resources anchored to that point in time.

For more detail about user or system queries, click a data point on one of the peaks. A Detail page opens to provide information about the queries in tabular format, including the query type, session ID, node name, query type, date, time, and the actual query that ran.
The bottom of the page indicates the number of queries it is showing on the current page, with Previous and Next buttons to navigate through the pages. You can sort the columns and export contents of the table to a file.

To return to the main Queries page, click <database name> Activity in the navigation bar.
Monitoring Internal Sessions

The Internal Sessions chart provides information about Vertica system activities, such as Tuple Mover and rebalance cluster operations, along with their corresponding resources, such as CPU/memory, networking, and disk I/O percent used.

Hover your cursor over a bar for more information. A dialog box appears and provides details.

Filtering Chart Results

You can filter what the chart displays by selecting options for the following components. As you filter, the Records Requested number changes:

- **Category**—Filter which internal session types (moveout, mergeout, rebalance cluster) appear in the graph. The number in parentheses indicates how many sessions are running on that operation.

- **Session duration**—Lists time, in milliseconds, for all sessions that appear on the graph. The minimum/maximum values on which you can filter (0 ms to \( n \) ms) represent the minimum/maximum elapsed times within all sessions currently displayed on the graph. After you choose a value, the chart refreshes to show only the internal sessions that were greater than or equal to the value you select.

- **Records requested**—Represents the total combined sessions for the Category and Session Duration filters.

Monitoring User Sessions

The User Sessions charts provide information about Vertica user activities for all user connections open to MC.

Choose User Sessions from the menu at the top of your database's Activity page to view these charts.
View Open Sessions

The Open Sessions tab displays a table of currently open sessions for each user. You can close a session or cancel a query on this tab by selecting that option from the Actions column.

Click any row to open a Session Details dialog that shows more extensive information about that session.
To configure the Open Sessions page display:

- Use the Sort Users button at the top right of the page to sort by user name or number of open sessions.
Use the Toggle Columns button at the top right of the page to select which columns to display. Each table displays session information by column, such as the session start time or the

![Session Table](image)

**View All User Sessions**

The All Sessions tab displays a history of all user sessions in a swim lane chart.
What Chart Colors Mean

Bars outlined with a dotted line are currently running sessions.

Sessions are divided into two colors, yellow and blue.

- Yellow bars represent user (system) sessions. If you click a yellow bar, MC opens a Detail page that shows all queries that ran or are still running within that session.

- Blue bars represent user requests (transactions within the session). If you click a blue bar in the graph, MC opens a Detail page that includes information for that query request only.

  When you hover your mouse over a transaction bar, a dialog box provides summary information about that request, such as which user ran the query, how long the transaction took to complete, or whether the transaction is still running.

Filter Chart Results

Extremely busy systems will show a lot of activity on the interface, perhaps more than you can interpret at a glance. You can filter chart results in several ways:
- **Zoom.** The context chart at the bottom of the page highlights in blue which section of the All Sessions chart you are viewing. Click and drag the blue box left or right to view earlier or later user sessions. Click and drag the edges of the blue box to zoom in or out.

- **Select fewer users.** Click the filter icon \( \text{-filter icon} \) at the top of the page. A menu of a menu of all available users appears below. Deselect users to exclude from the chart.

- **Change the session duration (how long a session took to run).** Click the Filter icon \( \text{filter icon} \) at the top of the page. The Filter sessions and queries by duration field appears below. Enter the minimum session length (in seconds) to display on the chart and click Update.

- **Specify a time frame.** Click the Calendar icon \( \text{calendar icon} \) at the top of the page to display the From and To fields. Using the fields, select the time frame to display in the chart and click Update.

---

**Monitoring System Memory Usage**

The Memory Usage chart shows how system memory is used on individual nodes over time. Information the chart displays is stored based on Data Collector retention policies, which a superuser can configure. See [Configuring Data Retention Policies](#).

The first time you access the Memory Usage chart, MC displays the first node in the cluster. MC remembers the node you last viewed and displays that node when you access the Activity page again. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart. The chart automatically refreshes every five minutes unless you disable the Auto Refresh option.

*Tip:* On busy systems, the Node list might cover part of the graph you want to see. You can move the list out of the way by dragging it to another area on the page.
Types of System Memory

The Memory Usage chart displays a stacking area for the following memory types:

- swap
- free
- fcache (file cache)
- buffer
- other (memory in use by all other processes running on the system besides the main Vertica process, such as the MC process or agents)
- Vertica
- rcache (Vertica ROS cache)
- catalog

When you hover over a data point, a dialog box displays percentage of memory used during that time period for the selected node.

Monitoring System Bottlenecks

The System Bottlenecks chart helps you quickly locate performance bottlenecks on a particular node. The first time you access the Activity page, MC displays the first node in the cluster. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart.
The System Bottlenecks chart reports what MC identifies as the most problematic resource during a given time interval. You can use this chart as a starting point for investigation.

How MC Gathers System Bottleneck Data

Every 15 minutes, MC takes the maximum percent values from various system resources and plots a single line with a data point for the component that used the highest resources at that point in time. When a different component uses the highest resources, MC displays a new data point and changes the line color to make the change in resources obvious. Very busy databases can cause frequent changes in the top resources consumed, so you might notice heavy chart activity.

In the following example, at 08:24 the maximum resources used changed from Disk I/O to CPU. The System Bottlenecks charts denote this with a change in line color from brown to green.
The Components MC Reports on

MC reports maximum percent values for the following system components:

- Average percent CPU usage
- Average percent memory usage
- Maximum percent disk I/O usage
- Percent data sent over the network (TX)
- Percent data received over the network (RX)

How MC Handles Conflicts in Resources

If MC encounters two metrics with the same maximum percent value, it displays one at random. If two metrics are very close in value, MC displays the higher of the two.

Monitoring User Query Phases

The User Query Phases chart provides information about the query execution phases that a query goes through before completion. Viewing this chart helps you identify at a glance queries possibly delayed because of resource contention.
Each bar, bound by a gray box, represents an individual query. Within a query, a different color represents each query phase. The chart does not show phases for queries with durations of less than 4 seconds. Blank spaces within a query represent waiting times, as in the image below.

Hover over a phase in the query for information on the phase type and duration.

The chart shows queries run over the last 15 minutes. The chart automatically refreshes every five minutes, unless you clear the Auto Refresh option in the toolbar.

**Filtering Chart Results**

You can filter what the chart displays by selecting options for the user running the query, minimum query duration, and minimum phase duration.
Viewing More Detail

To zoom in for detail, click-drag the mouse around a section of the chart. Click Reset Zoom, located at the top right corner of the chart, to restore the chart to its original view.

For more detail, click a query bar. The Detail page opens to provide information about the queries in tabular format, including the query type, session ID, node name, query type, date, time, the actual query that ran, and an option to run Explain Plan or profile the query. Click a table column header to sort the queries by that category.

To export the contents of the table to a file, click Export, located at the upper right of the page.

To return to the main Queries page, click Activity in the navigation bar.

Monitoring Table Utilization and Projections

The Table Utilization activity page helps you monitor tables and projections in your database by schema.

You can use the table utilization charts on this page to identify outliers among your tables, such as those that are large or overused. The projections summary can help identify if projections are evenly distributed across nodes.
Table Visualizations

To specify a schema to view, choose one from the Schemas menu at the top of the activity page. The summary of tables and projections in that schema appear on the page.

MC visualizes your available tables by schema in a table chart or as a treemap chart.

The Table chart provides flexible sorting to help you find tables you are interested in, especially if you have a large number of tables. You can use the Treemap chart for an at-a-glance graphical visualization of the relative size of your tables.

To choose how to visualize your table data, click Table or Map from the Show As menu. By default, MC displays the Table visualization. The current visualization is highlighted in blue in the Show As menu.

**Show as: **Table | Map

The Table chart, shown below, displays each table's row count, usage (percentage of time the table is queried), and a bar representing both row count and usage. The length of the bar indicates row count; a darker color indicates higher usage.

Hover over a bar in the chart to view details about that table, or click the table name to view its Table Details page.
In the Treemap visualization, tables are represented by boxes nested by size and colored by usage. Darker colors indicate higher table usage.

Depending on the number of tables in your database, the chart may be crowded. Use the Show Only filter at the top of the page to display only the largest or smallest 100 tables.

Hover over a table to view more details, or click to view its Table Details page.
The Table Details page displays information on the table's properties and columns, and about projections for the table. To the right, bar charts provide information, by node, on the amount of storage, number of deleted rows, and number of deleted vectors.

![Table Details Page Screenshot]

Note: If you have deleted table rows recently, Management Console may not display the most recent row count. MC updates the row count when mergeout occurs. See Mergeout.

**Projections Summary**

The projections summary displays information about projections by schema:

- **Total projections.**
- **Segmented projections,** the number of projections segmented across multiple nodes.
- **Unsegmented projections,** the number of projections that are not segmented across multiple nodes.
- **Projections Showing Distribution Skew,** the number of projections unevenly distributed across nodes. Tables with fewer than 1000 rows are not counted. Move the slider to configure filter by distribution skew percentage.
- **The number of projections with more than the specified number of containers per node.** Move the slider to configure the minimum number of containers.
Unsafe projections, the number of projections with a K-safety less than the database's K-safety.

Unused projections.

Click a projections number to view a list of the specified projections and their details.

Monitoring Running Queries

The Query Monitoring activity page displays the status of recent and currently running queries, and resource information by node and user. From this page, you can profile a query or cancel a running query.

Use this page to check the status of your queries, and to quickly cancel running or queued queries to free up system resources. This page can help you identify where resources are being used, and which queries, users, or nodes are using the most resources.

The Query Monitoring page includes four tables, displayed as tabs:

- Running queries
- Queued queries
- Completed queries
- Failed queries
From the Actions column you can:

- **Cancel.** Cancel a running or queued query.
- **Close session.** Close a session for a running or queued query.
- **Explain.** Open the Query Plan page for any query.
- **Profile.** Profile any query on the Query Plan page.

The four bar charts at the bottom of the page display aggregate query usage by node or user. Hover over a bar with your cursor to see its value. When sorted by value, the left-most bar in each chart represents the node or user with the highest value.

The Query Monitoring page refreshes every 20 seconds by default. To change the refresh interval, click the Page Settings button in the upper-right corner of the page. A dialog appears. Type the new refresh interval, in milliseconds, in the text box.

Filtering Chart Results

Use the search field below each column title to narrow down your chart results. (For example, if you enter the text `SELECT product_description` in the Search Queries field and select a specific node in the Initiator Node column, the chart returns only queries which both contain that text and were initiated on the node you specified.)

Click a column title to sort the order of the queries by that category.

There may be a large number of results for Completed and Failed Queries. Use the Customize section at the top of these two tabs to further filter your chart results. For either tab, you can select a custom date and time range for your results.
In the Completed Queries tab, click Data to enter additional query information to filter based on any of the following fields:

- User
- Request
- Request Duration
- Node
- Request label

**Viewing More Details**

Click a query to view the entire query.

In the Failed Queries chart, click the plus (+) icon next to a failed query to see the failure details for each node involved in the query's execution.

To export the data in one of the Query Monitoring tables, click the table's tab, then click the Export (⬇️) button in the upper-right corner of the page. The browser downloads the data for that chart in a .dat file. The data exported includes columns that may not be visible in the MC, including the minimum and maximum values for Memory, Thread Count, and Open File Handles.
Monitoring Catalog Memory

The Catalog Memory activity page displays the catalog memory for each node. Use this page to check for sudden changes in catalog memory, or discrepancies in memory distribution across your nodes.

The Catalog Memory page displays:

- A node details table. The table lists the details of each node in the database, including their current catalog memory and total memory usage.
- A catalog memory chart. A line graph visualization of each node's catalog memory usage over time. Each line represents a node. The color legend at the bottom of the chart indicates the color of each node's line.

In the image below, catalog memory begins at 0GB for all three nodes. Over the next twenty minutes, catalog memory increases to 0.04GB in the second node (orange), then the first node (cyan), and finally in the third node (dark blue). Starting at 16:55, note that the three overlapping node lines appear as one line when all three nodes have the same catalog memory.

Filtering Chart Results

If you have many nodes in your database, you may want to display only certain nodes in the catalog memory chart. You can remove nodes from the chart in two ways:
• Deselect the node's check box in the node details table.

• Deselect the node in the color legend below the chart.

**Viewing More Details**

Hover over any line in the chart to view the time, node name, and catalog size.

At the bottom of the chart is a summary bar that shows a quick overview of the catalog memory over time. Move the sliders on either side of the chart to zoom in on a specific time frame in the chart. When zoomed in, you can use the scrollbar to move forward or backward in time.

**Monitoring Depot Activity in MC**

When your database is in Eon Mode Beta, you can use the Depot Activity page to monitor query activity, churn, and settings for your database's depot. To view this page, click the Activity tab at the bottom of your database's Overview page, then select Depot Activity Monitoring from the chart list. This page is not available when the database is in Enterprise Mode. See *Using Eon Mode Beta* for more about Vertica database modes.

The depot improves query performance by preventing your queries from having to go to communal storage every time. Instead the queries access the depot, which contains local copies of your data.

The page contains three sections:

• Depot Usage and Settings

• Completed Depot Fetches and Evictions
User Query Activity by Storage Location

Monitor Depot Usage and Settings

The Depot Usage and Settings module, on the left side of the page, lists:

- Depot usage. Click the View More link next to Details by Nodes to display Depot usage for individual nodes. You can also access the Storage View page by clicking the View More on Database Storage link.

- Depot configuration settings.

- Retention limits for monitoring depot activity.

The retention policies determine how long Vertica retains the depot monitoring information the depot activity charts display. Consider increasing retention limits, such as that of Depot Reads, to display a longer history of activity.
Monitor Query Activity by Storage Location

The User Query Activity by Storage Location chart indicates whether your queries access the depot or communal storage. For optimal performance, the majority of queries should access your depot, which is local.

When the depot size is more limited, queries must more often access communal storage to retrieve all the necessary data. A more limited depot size may be necessary due to storage costs, but accessing communal storage can impact query performance.

For more details:

- Hover over the line at any point in time to see details about the number of queries that ran and how many nodes they ran on.
- Click the line to view the Query Details page, which provides information about every query that ran.

Monitor Depot Fetches and Evictions

The Completed Depot Fetches and Evictions chart shows when the depot evicts data, or fetches data from communal storage.

Use this to monitor churn (how many files are getting pushed out of the depot), and how up-to-date your depot's data is. Churn occurs when, with a limited depot size, the depot must begin to fetch and evict data more frequently. If you observe that queries are consistently slower due to accessing communal storage, and notice the depot keeps experiences frequent churn, you might benefit from increasing your depot's size limit.

The line graph visualization displays depot fetches in green, and depot evictions in red.

Monitoring Resource Pools

Management Console allows database administrators to monitor and configure resource pools through the Activity and Configuration pages. These pages help you manage workloads by providing visual representations of resource usage as well as resource pool configuration options.
Monitoring Resource Pools Charts

You can monitor your resource pools using the Resource Pools Monitoring charts, accessible through the Management Console Activity page.

Select a resource pool to view using the Resource Pool menu, located in the leftmost sidebar. In the sidebar, Current Usage Activity displays the pool’s real-time statistics.

Monitor the selected resource pool using the following charts, which display the pool’s historic data:

- **Resource Usages in Pool**: Shows the historically averaged acquired memory usage by each pool across all nodes. The graph uses two y-axes, one that shows memory size, and a second that shows the total number of running queries. Data is collected every hour. Hover over a data point for a summary of the memory usage at that specific point.

- **Memory Usage in Node**: Shows the historically acquired memory usages by all pools across all nodes. Data is collected every hour. Hover over a data point for a summary of the memory usage at that specific point.

- **Average Query Execution and Query Time in Pool**: Shows the averaged query queue time plotted against the query execution time by each pool across all nodes. Data is collected every minute. Hover over data to get the average query execution and queue time in the specified pool. Click a data point to show detailed individual query information.
- **Resource Rejections in Pool**: Shows the historical total number of resource requests that were rejected in each pool across all nodes. Data is collected every hour. Click a data point to show rejection details and reasons in a pop-up window.

## Configuring Resource Pools in MC

Database administrators can view information about resource pool parameters and make changes to existing parameters through the Management Console Configuration page. You can also create and remove new resource pools, assign resource pool users, and assign cascading pools.

See [Configuring Resource Pools in Management Console](#).

## Permissions

Only the database administrator can monitor and configure resource pools in Management Console.

## See Also

- [Configuring Resource Pools in Management Console](#)
- [Using Queries to Monitor Resource Pool Size and Usage](#)

## Configuring Resource Pools in Management Console

Database administrators can view information about resource pool parameters and make changes to existing parameters in MC through the Resource Pools Configuration page. You can also create and remove new resource pools, assign resource pool users, and assign cascading pools.


You can also access the Configuration page from the Resource Pools Monitoring chart, accessible through the Management Console Activity page. Click the tools icon on the top of the leftmost sidebar.
Permissions for Monitoring and Configuring Resource Pools

Only the database administrator can monitor and configure resource pools in Management Console.

Modify Resource Pool Parameters


2. Use the parameter fields to view and modify parameters for the resource pool. Hover your cursor in the parameter field to display information about that parameter.

3. Click Apply to save your changes. A success message appears.

Modify Resource Pool Users

To add or remove resource pool users:


3. The dialog displays users assigned to the resource pool in the Current Pool Users list. The Other Users list displays all other resource pool users are displayed, along with the pool to which they are currently assigned.
a. To add users to the resource pool: Select the desired users from Other Users list and click Add.

b. To remove users from the resource pool: Select the users to be removed from the Current Pool Users list and click Remove.

4. Click Apply to save your changes. A success message appears.

Create and Remove Resource Pools

Database administrators can use MC to create resource pools and assign resource pool users, and remove user-generated resource pools.

To create a resource pool:


2. Enter the new resource pool's parameters in the fields.

3. Click Create Pool. A success message appears.

To remove a resource pool:

1. First, remove all users from the resource pool to be deleted. This can be done on the Resource Pool Configuration Page.

2. When all users have been removed from the resource pool, choose the resource pool from the Resource Pools field on the Resource Pool Configuration Page. Parameter fields for that resource pool appear.


4. Click OK in the Confirm dialog. A success message appears.

See Also

- CREATE RESOURCE POOL
- Monitoring Resource Pools
- Using Queries to Monitor Resource Pool Size and Usage
Monitoring Database Messages in MC

As Management Console monitors your database, it periodically checks your system's health and performance. MC then generates messages to alert you about the state of your system. You can view and manage these messages in the Message Center.

Getting New Message Notifications

The Messages icon appears in the top-right of any database-specific page on MC (such as the Overview page or Activity page). The icon displays a new messages badge. The badge displays how many highest priority messages you have received recently.

The messages badge displays the color and letter of that priority:

- Red (H): High Priority
- Orange (N): Needs Attention
- Blue (I): Informational

In addition, the Thresholds Notification widget on the Overview page displays a summary of alerts about exceeded thresholds. Only high-priority threshold alerts appear in the Thresholds Notification widget. You can set the priority of alerts for different thresholds in the Thresholds tab on your database's Settings page.

Viewing Message Center

You can look at a preview of your most recent messages without navigating away from your current page. Click the Message Center icon in the top-right corner of Management Console to view the notification menu. The menu appears as a drop-down preview of your inbox. From this menu, you can delete, archive, or mark your messages as read.
You can access the Message Center in three ways:

- Click the Message Center button on the MC Home Page. The Message Center displays messages about the most recent database you have viewed.

**Manage Information**

- **Existing Infrastructure**: Manage databases and clusters
- **Diagnostics**: View diagnostics and support information
- **MC Settings**: Manage application and user settings
- **Message Center**: Go to the database message center
Click the Messages icon in the top-right of any database-specific page on MC. (For example, the icon appears on the Overview and Activity pages). A menu of your recent messages appears. Click Message Center in the top-right of the menu to reach the Message Center for the database you most recently viewed.

![Message Center](image)

Click the Messages widget on the right side of the Overview page.

![Messages widget](image)

**Filtering Recent Messages**

By default, Message Center displays up to 600 of a database's most recent messages from the past week.
To see more messages:

1. Click Retrieve Older Messages in the top-right of the page.

2. Use the drop-down From and To calendars to select a time range from which to retrieve older messages. You can specify days, hours, and minutes.

3. Click Search to display the messages from the specified time frame.

Depending on your database, you may have received hundreds of recent messages in the past seven days. To help you find the messages most relevant to you, Message Center provides several ways to filter recent messages:

- **By search term:** Use the search field at the top of the page to filter messages by any search term, such as description or date.

- **By database:** View a specific database’s messages by choosing it from the Showing: [database] drop-down list at the top of the page. Message Center’s default view lists up to 600 of the most recent messages for each database MC monitors.

- **By time period:** Message Center groups your recent messages by Today, Yesterday, and This Week. Click any of these headings to collapse that section of messages.

- **By message priority:** Messages are categorized as High Priority, Needs Attention, or Informational. The number of messages of each priority level appears at the top-right of the page. By default, all priority levels are displayed in the inbox. To filter messages of one priority level out of your current view, click a message count value at the top of the inbox to deselect it.

- **By threshold messages:** MC generates messages indicating when certain thresholds are exceeded in your database. To view only messages related to exceeded thresholds, click the
Thresholds Messages tab on the left hand side of the page. To modify these alert thresholds, go to the Thresholds tab, which appears in your database’s Settings page.

By default, Message Center displays up to 600 recent entries per database. You can adjust the maximum limit of messages the Message Center displays in the /opt/vconsole/config/console.properties file. To adjust the limit, change the number in the following line:

```
messageCenter.maxEntries=600
```

Managing and Archiving Messages

Message Center archives all messages you mark as read. To mark messages as read or unread, select a message or multiple messages in the Message Center and click Mark Read or Mark Unread.

You can also perform the following tasks:

- See an archived message—Click Archived Messages on the right side of the Message Center. Then, use the From and To fields at the top of the page to select a date range from which to display archived messages.

- Sort archived messages by Type, Database Name, Description, or Date—Click any filter at the top of the Message Center.

- Delete messages permanently—If you prefer to delete messages, rather than archiving them, select the messages to delete. Then, click Delete Msgs.

Viewing Message Details

Within the Message Center, click the plus (+) symbol next to a message to get more information about the issue. You can also query the V_MONITOR.EVENT_CONFIGURATIONS table to get information about events. See the SQL Reference Manual for details.

Message Priority Levels

MC sorts messages by priority level and prioritizes them with color codes:
- High Priority (Red)
- Needs Attention (Orange)
- Informational (Blue)

At the top of your inbox, the Message Center displays the number of messages that fall within each level of priority.

Click any of the values to deselect them. This filters that type of message out of your current inbox view. For example, suppose you want to view the High Priority messages in your inbox. To remove lower-priority messages, you can click the message count values for the categories labeled Needs Attention and Informational. Doing so removes them from your current view.

Messages in the inbox are additionally labeled with one of seven subcategories, according to priority:

- High Priority: Emergency (0), Alert (1)
- Need Attention: Critical (2), Error (3)
- Informational: Warning (4), Info (5), Notice (6)

See Also

- Customizing Threshold-Based Notifications
- Exporting MC-managed Database Messages and Logs

Message Types

MC generates both default notifications and customizable threshold notifications.

Threshold Notifications

MC generates threshold-based notifications when the database exceeds specified limits on the following thresholds.
- License Status
  - License Usage

- System Health
  - Node Health
    - Node CPU
    - Node Memory
    - Node Disk Usage
    - Node Disk I/O
    - Node CPU I/O Wait
    - Node Reboot Rate
    - Node Catalog Memory
    - Node State Change
  - Network Health
    - Network I/O Error

- System Performance
  - Query
    - Queued Query Number
    - Failed Query Number
    - Spilled Query Number
    - Retried Query Number
    - Query Running Time
  - Resource Pool
    - Queries Reaching the Max Allowed Execution Time
    - Queries With Resource Rejections
    - Ended Query With Queue Time Exceeding Limit
    - Ended Query With Run Time Exceeding Limit
MC provides default settings for these notifications. You can customize threshold notification settings in the Thresholds tab in the database Settings page. Threshold notifications appear in the Message Center. High-priority threshold alerts also appear on the Overview Page. See Customizing Threshold-Based Notifications for how to prioritize alerts.

Default Notifications

By default, MC also generates messages about the database that appear only in the Message Center. You might receive messages about the following database-related conditions:

- Low disk space
- Read-only file system
- Loss of K-safety
- Current fault tolerance at critical level
- Too many ROS containers
- WOS overflow
- Change in node state
- Recovery error
- Recovery failure
- Recovery lock error
- Recovery projection retrieval error
- Refresh error
- Refresh lock error
- Workload Analyzer operations
- Tuple Mover error
- Timer service task error
- Last Good Epoch (LGE) lag
Customizing Threshold-Based Notifications

Management Console can generate notifications when your database exceeds threshold limits that you specify. You can configure threshold notifications per database in the Thresholds tab, which appears on your database's Settings page.

You can enable and modify message thresholds for each database that MC monitors. MC then generates a message indicating when a threshold is exceeded. For example, you can set the threshold for node disk usage to a 20% minimum and 80% maximum. As MC monitors node disk usage, it notifies you when any nodes exceed those minimum or maximum thresholds.

MC categorizes customizable message thresholds by license use, system health, and system performance. See Message Types for a list of the customizable thresholds available in MC.


Configuring Settings for Thresholds

Message thresholds have the following configurable settings:

- Enabling and disabling threshold-specific messages
- Threshold values
- Check time interval
- Alert Priority
- Email Destination
- Email Interval

Configure your notification settings, and click Apply to save your changes. If a message threshold has not been set previously, MC displays the default threshold for that setting.
Setting Priorities

You can set a notification to one of three priorities. If a threshold value is exceeded, MC:

- Priority 1 — Sends you an email notification, displays a message in the Overview page, and creates an message in the Message Center.
- Priority 2 — Displays a message in the Overview page, and creates a message in the Message Center.
- Priority 3 — Creates a message in the Message Center.

Setting Email Notifications

In order for MC to send email notifications, you must first provide MC with SMTP server settings. See Set Up Email.

When you set a threshold to Priority 1, MC can send email notifications to subscribed users.

To subscribe users to a threshold email message:

1. Set the threshold’s Alert Priority field to Priority 1: Overview and Email. The Email Destination and Email Interval fields appear.
2. Click the browsing icon next to the Email Destination field. The Subscriber List dialog box opens.
3. You can add emails in one of two ways:
   - Select from the list of existing MC users with associated email addresses. (See Managing MC Users for how to add email addresses to user profiles.)
   - Provide an additional email address in the Entering New Email field. Click the plus (+) icon next to the field to add the address.
4. Click OK.
5. Select an option from the Email Interval field to choose how frequently users receive email notifications about the threshold if it is exceeded.
6. Click Apply at the top-right of the page to save your email subscriber settings.
See Also

- Message Types
- Set Up Email
- Managing MC Users

Set Up Email

Management Console can generate email alerts about high-priority database thresholds. To receive email alerts, you must first configure your SMTP settings in MC.

You must be an administrator to provide SMTP settings.

To set up MC to send email:

1. Select the Email Gateway tab on the MC Settings page.

2. Provide the following information about your SMTP server:
   - SMTP Server (Hostname). Maximum length is 255 characters. You can enter either the hostname or IP address.
   - SMTP server port.
   - Session Type (SSL or TLS).
   - SMTP Username.
   - SMTP Password.
   - Originating Email Alias. MC sends alerts from the email address you provide.

3. Click Test at the top of the page. MC validates your SMTP settings and sends a test email to the inbox of the email alias you provided.

4. Verify that you successfully received the test email.

5. Click Apply at the top-right of the page to save the settings.
With email settings completed, you can configure MC to send high-priority threshold alerts through email. See Customizing Threshold-Based Notifications.

Searching Database Messages Managed by MC

The Management Console Message Center displays the 10,000 most recent database messages, starting with the most recent record.

If more than 600 recent messages exist for a database, MC returns a message letting you know that only the most recent 600 entries are shown, so you have the option to filter.

Changing Message Search Criteria

You can use the search field at the top right of the Message Center to filter your messages by any search term, such as database name, description, or date.

Additionally, click on any filter at the top of the Message Center to sort message by Type, Database Name, Description, or Date. To further filter messages, you can select a database from the Database Name drop down list, type a keyword in the Description filter, or enter a range of dates in the Date filter.

Retrieving Additional Messages

If more than 10,000 messages exist, older messages that occurred before the 10,000 messages do not appear in the Message Center by default. You can click Retrieve Additional Alerts to filter for these older messages.

To specify which messages to retrieve, click Retrieve Additional Alerts. The Search for Messages dialog appears. Choose a date range, one or more message types, or one or more message types within a specific date range. Click OK to view the messages in that range.
You can adjust the maximum limit of recent messages the Message Center displays in the /opt/vconsole/config/console.properties file. To adjust the limit, change the number in the following line:

```
messageCenter.maxEntries=5000
```

### Specifying Date Range Searches

For date range searches, MC starts at the beginning of the time range and either returns all messages up to the specified end time or 10,000 messages, whichever comes first. You can filter message searches on the following date ranges:

- Any date-to-date period, including hour and minute
- Any time period up to now (forward searches)
- Any time period before now (backward searches)

After you specify a date range, click **Done** to close the calendar, and then click **OK** to see the results in the Message Center.

### Exporting MC-managed Database Messages and Logs

You can export the contents of database messages, log details, query details, and MC user activity to a file.
Information comes directly from the MC interface. This means that if the last five minutes of `vertica.log` information displays on the interface, you can save that five minutes of data to a file, not the entire log. When you filter messages or logs, MC exports only the filtered results.

Depending on how you set your browser preferences, when you export messages you can view the output immediately or specify a location to save the file. System-generated file names include a timestamp for uniqueness.

The following table shows, by record type, the MC pages that contain content you can export, the name of the system-generated file, and what that file's output contains:

<table>
<thead>
<tr>
<th>Message type</th>
<th>Where you can export on MC</th>
<th>System-generated filename</th>
<th>Contents of exported file</th>
</tr>
</thead>
<tbody>
<tr>
<td>All db-related message types</td>
<td>Message Center page</td>
<td>vertica-alerts-&lt;timestamp&gt;.csv</td>
<td>Exports messages in the Message Center to a <code>.csv</code> file. Message contents are saved under the following headings:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Create time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Severity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Summary (of message)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Description (more details)</td>
</tr>
<tr>
<td>MC log files</td>
<td>Diagnostics page</td>
<td>mconsole-&lt;timestamp&gt;.log</td>
<td>Exports MC log search results from MC to a <code>.log</code> file under the following headings:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Type (message severity)</td>
</tr>
<tr>
<td>Message type</td>
<td>Where you can export on MC</td>
<td>System-generated filename</td>
<td>Contents of exported file</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
</tbody>
</table>
| Vertica logs | Manage page                 | `vertica-vertica-<db>-<timestamp>.log` | - Component (such as TM, Txn, Recover, and so on)  
- Message |
|              | Double-click any node to get to the details and then click the VerticaLog tab |                           |                          |
| Agent logs   | Manage page                 | `vertica-agent-<db>-<timestamp>.log` | - Component (such as TM, Txn, Recover, and so on)  
- Message |
<p>|              | Click any node to get to the details and then click the AgentTools Log tab |                           |                          |</p>
<table>
<thead>
<tr>
<th>Message type</th>
<th>Where you can export on MC</th>
<th>System-generated filename</th>
<th>Contents of exported file</th>
</tr>
</thead>
</table>
| Query details     | **Activity page**  
Click any query spike in the Queries graph to get to the Detail page.  
| vertica-querydetails-<db>-<timestamp>.dat |                                                                                           | Exports query details for the database between <timestamp> and <timestamp> as a tab-delimited .dat file. Content is saved under the following headings:  
|                   |                                                                                           | • Query type  
• Session ID  
• Node name  
• Started  
• Elapsed  
• User name  
• Request/Query | |
| MC user activity  | **Diagnostics page**  
Click the Audit Log task  
| vertica_audit<timestamp>.csv  |                                                                                           | Exports MC user-activity results to a .csv file. Content is saved under the following headings:  
|                   |                                                                                           | • Time  
• MC User  
• Resource  
• Target User  
• Client IP | |
Monitoring MC User Activity Using Audit Log

When an MC user makes changes on the MC interface, whether to an MC-managed database or to the MC itself, their action generates a log entry that records a timestamp, the MC user name, the database and client host (if applicable), and the operation the user performed. You monitor user activity on the Diagnostics > Audit Log page.

MC records the following types of user operations:

- User log-on/log-off activities
- Database creation
- Database connection through the console interface
- Start/stop a database
- Remove a database from the console view
- Drop a database
- Database rebalance across the cluster
- License activity views on a database, as well as new license uploads
- Workload Analyzer views on a database
- Database password changes
- Database settings changes (individual settings are tracked in the audit record)
- Syncing the database with the cluster (who clicked Sync on grid view)
- Query detail viewings of a database
- Closing sessions
- Node changes (add, start, stop, replace)
- User management (add, edit, enable, disable, delete)
- LDAP authentication (enable/disable)
- Management Console setting changes (individual settings are tracked in the audit record)
- SSL certificate uploads
- Message deletion and number deleted
- Console restart from the browser interface
- Factory reset from the browser interface

**Background Cleanup of Audit Records**

An internal MC job starts every day and, if required, clears audit records that exceed a specified timeframe and size. The default is 90 days and 2K in log size. MC clears whichever limit is first reached.

You can adjust the time and size limits by editing the following lines in the `/opt/vconsole/config/console.properties` file:

```plaintext
vertica.audit.maxDays=90
vertica.audit.maxRecords=2000
```

**Filter and Export Results**

You can manipulate the output of the audit log by sorting column headings, scrolling through the log, refining your search to a specific date/time and you can export audit contents to a file.

If you want to export the log, see Exporting the User Audit Log.

**If You Perform a Factory Reset**

If you perform a factory reset on MC's Diagnostics page (restore it to its pre-configured state), MC prompts you to export audit records before the reset occurs.
Monitoring External Data Sources in Management Console

By default, Management Console monitors a database using information from that database's Data Collector (DC) tables. MC can also monitor DC tables you have copied into Vertica tables, locally or remotely.

MC administrators provide mappings to local schemas or to an external database containing the corresponding DC data. MC can then render its charts and graphs from the new repository instead of from local DC tables. This offers the benefit of loading larger sets of data faster in MC, and retaining historical data long term.

Note: MC also offers External Monitoring, which allows you to set up a Vertica storage database through the MC interface, then use Kafka to stream your data to the storage database. You can use the Data Source mapping process below if you prefer to set up your own alternative data source, or do not plan to use Kafka streaming.

Map an Alternative Data Source

1. On the MC Settings page, navigate to the Data Source tab.

2. Select the database for which you are creating the data source mapping.

3. Choose the database user for which you want to create the mapping.

4. Set Repository Location to Local or Remote.

5. If Remote is selected, provide JDBC connection parameters for the remote database repository. Click Validate Connection Properties to confirm a successful connection.

6. Enter the schema mappings for v_internal and v_catalog. MC does not support mapping the v_monitor schema.

7. Input your table mappings in one of the following ways:
   - Click Auto Discover. MC retrieves the table mappings based on the database and schema mappings you provided.
   - Click Manual Entry. Manually input table mappings.
- Click Load Configurations. If you previously saved a data source configuration for the database in a file, import the file to use that configuration for the currently selected user.

8. Optionally, click Save Configurations to export this configuration file. You can create a mapping for another database user with this configuration file later.

9. Click Apply to save and apply your configuration settings.

Reports Using Unmapped Schemas

If a report in MC needs to access a locally stored schema or table that is unmapped, MC includes information from the local DC tables for that schema to complete the report.

For remote configurations, if a report depends on an unmapped schema or table, the entire report is run against the local DC tables. If the remote database is down when MC attempts to run a report against it, MC reruns the report against the local database.

When the MC runs a report, it records missing mappings in the MC log under the INFO severity level.
Loading Data Using MC

You can use the Data Load Activity page in Management Console to import data from Amazon S3 storage to an existing table in Vertica.

MC uses the Vertica library for Amazon Web Services (AWS) to import data directly from Amazon S3 storage to Vertica. You do not need to use any third-party scripts or programs. When you run a loading job, Vertica appends rows to the target table you provide. If the job fails, or you cancel the job, Vertica commits no rows to the target table.

When you view your load history on the Instance tab, loading jobs initiated in Management Console using Amazon S3 have the name MC_S3_Load in the Stream Name column.

For more information about loading data from Amazon S3 storage to Vertica, see Bulk Loading and Exporting Data From Amazon S3.

Prerequisites

To use the Load feature in Management Console, you must first have:

- Access to an Amazon S3 storage account.

- An installed version of Vertica 7.2.2 or later, which includes the Vertica library for Amazon Web Services.

- An existing table in your Vertica database to which you can copy your data. You must be the owner of the table.

Create a Loading Job

To load data from an Amazon S3 bucket to an existing table in your target database:

1. On your target database's Management Console (MC) dashboard, click the Load tab at the bottom of the page to view the Data Load Activity page.

2. Click the Instance tab.

3. Click New S3 Data Load at the top-right of the tab. The Create New Amazon S3 Loading Job dialog box opens.
4. Enter your AWS account credentials and your target location information in the required fields, which are indicated by asterisks (*). Use the format S3:// for the bucket name.

5. (Optional) Specify additional options by completing the following fields:
   - Direct
   - COPY Parameters
   - Capture rejected data in a table
   - Reject max

   For more about using these fields, see About Configuring a Data Load from S3.

**Cancel an Initiated Loading Job**

If a loading job is in progress, you can cancel it using the Cancel option in the Load History tab's Cancel column. Click Cancel to cancel the loading job. When you cancel a job, Vertica rolls back all rows and does not commit any data to the target table.

<table>
<thead>
<tr>
<th>Status</th>
<th>Time Started</th>
<th>User</th>
<th>Schema Name</th>
<th>Table Name</th>
<th>Execution Time (ms)</th>
<th>Accepted Rows</th>
<th>Rejected Rows</th>
<th>Stream Name</th>
<th>Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Mar 28, 2016 12:00</td>
<td>v1db_foo</td>
<td>vs_designs</td>
<td></td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td></td>
<td>Cancel</td>
</tr>
<tr>
<td>Success</td>
<td>Mar 28, 2016 12:00</td>
<td>natalia</td>
<td>store</td>
<td>store_orders_fact</td>
<td>110707</td>
<td>299994</td>
<td>Details</td>
<td>MC_S3_Load</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>Mar 28, 2016 12:00</td>
<td>natalia</td>
<td>store</td>
<td>store_orders_fact</td>
<td>581</td>
<td>0</td>
<td>Details</td>
<td>MC_S3_Load</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>Mar 28, 2016 12:00</td>
<td>natalia</td>
<td>store</td>
<td>store_orders_fact</td>
<td>1299</td>
<td>11</td>
<td>N/A</td>
<td>S2_Data_load</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td>Mar 28, 2016 12:00</td>
<td>natalia</td>
<td>store</td>
<td>store_orders_fact</td>
<td>2</td>
<td>0</td>
<td>N/A</td>
<td>S3_Data_load</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>Mar 25, 2016 3:00</td>
<td>v1db_foo</td>
<td>store</td>
<td>store_orders_fact</td>
<td>17590</td>
<td>300000</td>
<td>Details</td>
<td>MC_S3_Load</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

- Viewing Load History
- About Configuring a Data Load from S3
- Integrating with Apache Kafka
- Bulk Loading and Exporting Data From Amazon S3
- Loading Data From Amazon S3
About Configuring a Data Load from S3

When you create a new S3 Data Load using MC, you have the option of further configuring the job. You can optionally specify the following:

Load Data into ROS or WOS

You can use the Direct field to specify whether to load the data directly into ROS (Read Optimized Store) containers. This is the default setting, and best for large loads of 100MB or more. For smaller loads, deselect this option to load the data into WOS (Write Optimized Store).

Add COPY Parameters

MC performs the loading job using a COPY statement. You can use the COPY Parameters field to further configure the COPY statement, or leave it blank. Only parameters used after the SOURCE parameter in a COPY statement are valid in this field.

Note that if you use the FILTER and PARSER parameters, they must appear in that order and precede any other parameters you use in the COPY Parameters field.

Specify EXCEPTIONS only if you set the Capture rejected data in a table field to No.

In the following example, you could use the DELIMITER and SKIP parameters in the COPY parameters field to separate columns with a comma and skip 1 record of input data.

```
DELIMITER ',' SKIP 1
```

To add comments in the COPY statement using this field, begin your comment with a forward slash followed by an asterisk (/*) and end it with an asterisk followed by a forward slash (*/). Using a double hyphen (--) does not work in this field.

If COPY rejects the maximum number of rows, Vertica rolls back the target table rows without committing any data.
Capture Rejected Data in a Table

Use the Capture rejected data in a table field to create a rejected data table. A rejected data table allows you to view details about rejections through MC. If you set this field to Yes, you will be able to view rejected row data in the Load History tab.

You must have CREATE privilege on the schema if the table doesn't already exist. When you invoke multiple load processes for the same target table, MC appends all rejections data to the same table. MC generates a table using the following naming convention: \textit{schema.s3\_load\_rejections\_target-table-name}.

For more information about capturing rejected data, see Saving Rejected Data To a Table.

Capturing Rejected Data in a Table

Use the Capture rejected data in a table field to create a rejected data table. A rejected data table allows you to view details about rejections through MC.

You must have CREATE privilege on the schema if the table doesn't already exist. When you invoke multiple load processes for the same target table, MC appends all rejections data to the same table. MC generates a table using the following naming convention: \textit{schema.s3\_load\_rejections\_target-table-name}.

For more information about capturing rejected data, see Saving Rejected Data To a Table.

Set a Rejected Records Maximum

Use the Reject Max field to specify the maximum number of records that can be rejected before the load fails. If COPY rejects the maximum number of rows, Vertica rolls back the target table rows without committing any data.

Configuration Options

To further configure the loading job, you can use the following optional fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Use the Direct option to indicate whether to load the data into WOS or ROS.</td>
</tr>
<tr>
<td>Field</td>
<td>Details</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Selected (Default):</strong> The data loads directly into <strong>ROS (Read Optimized Store)</strong> containers. Use this for large bulk loads (100MB or more).</td>
</tr>
<tr>
<td></td>
<td><strong>Deselected:</strong> Data loads into <strong>WOS (Write Optimized Store)</strong>. Use this for smaller bulk loads.</td>
</tr>
<tr>
<td>COPY Parameters</td>
<td>Use the COPY Parameters field to further configure the COPY statement that will load your data.</td>
</tr>
<tr>
<td>Valid values:</td>
<td></td>
</tr>
<tr>
<td>FILTER</td>
<td></td>
</tr>
<tr>
<td>PARSER</td>
<td></td>
</tr>
<tr>
<td>DELIMITER</td>
<td></td>
</tr>
<tr>
<td>TRAILING NULLCOLS</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>ESCAPE (NO ESCAPE)</td>
<td></td>
</tr>
<tr>
<td>ENCLOSED</td>
<td></td>
</tr>
<tr>
<td>RECORD TERMINATOR</td>
<td></td>
</tr>
<tr>
<td>SKIP, SKIP BYTES</td>
<td></td>
</tr>
<tr>
<td>TRIM 'byte'</td>
<td></td>
</tr>
<tr>
<td>EXCEPTIONS 'path'</td>
<td></td>
</tr>
<tr>
<td>ENFORCELENGTH</td>
<td></td>
</tr>
<tr>
<td>ABORT ON ERROR</td>
<td></td>
</tr>
<tr>
<td>STORAGE</td>
<td></td>
</tr>
<tr>
<td>For more information about COPY syntax, see <strong>COPY and COPY Parameters</strong>.</td>
<td></td>
</tr>
<tr>
<td>Capture rejected data in a table</td>
<td>Use this field to create a rejected data table.</td>
</tr>
<tr>
<td>Yes: MC generates a rejected data table.</td>
<td></td>
</tr>
</tbody>
</table>
### Field Details

<table>
<thead>
<tr>
<th>Field</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>MC does not save rejected rows to a table. For more information, see Saving Rejected Data To a Table.</td>
</tr>
<tr>
<td>Reject max</td>
<td>Use this field to specify the maximum number of records that can be rejected before the load fails.</td>
</tr>
</tbody>
</table>

- **Loading Data Using MC**
- **Viewing Load History**
- **Integrating with Apache Kafka**
- **Bulk Loading and Exporting Data From Amazon S3**
- **Loading Data From Amazon S3**
- **COPY**
- **COPY Parameters**

## Viewing Load History

You can view a history of all your continuous and instance loading jobs in Vertica on the Data Load Activity page.

- **Continuous jobs**: Loading jobs that continuously monitor a source and stream data from the source.

- **Instance jobs**: Loading jobs that batch load from a source. Instance jobs are of a fixed length and shorter-term than continuous loads.

## View Continuous Loads

The Continuous tab on the Data Load Activity page displays history of your database’s continuous loading jobs. For example, you can see loading jobs you create using the Vertica integration with Kafka (see Integrating with Apache Kafka). Additionally, if you enable the MC extended monitoring feature, the Continuous tab displays the continuous jobs that stream data from your monitored database to a storage database. (See Extended Monitoring for more on how MC can use Kafka to monitor databases externally.)
Use the Continuous tab to view details about continuous jobs, such as their source, target tables, and other microbatch configuration details.

If extended monitoring is enabled, jobs streaming to the MC storage database show mc_dc_kafka_config as the scheduler name. Deselect Show MC data collector monitoring streams at the top of the tab to remove these jobs from the display.

In the Continuous tab, click the labels in the Scheduler, Microbatch, and Errors Last Hour to view additional details about those loading jobs.

For more on continuous data streaming terminology, see Data Streaming Integration Terms.

View Load Instances

In the Instance tab, you can see a history of your database's one-time loading jobs. For example, you can view instance jobs you created using the COPY command in vsql (see COPY), or instance jobs you created in MC to copy data from an Amazon S3 bucket. (For more about initiating loading jobs in MC, see Loading Data Using MC.)

In the Instance tab, click the labels in the Status column and Rejected Rows column to view more details about completed jobs. For more about rejected rows, see Capturing Load Rejections and Exceptions.
The number of load history results on the Instance tab depends on the Data Collector retention policy for Requests Issued and Requests Completed. To change the retention policy, see Configuring Data Retention Policies.

See Also

- Loading Data Using MC
- Integrating with Apache Kafka
- Integrating with Apache Kafka
- Bulk Loading and Exporting Data From Amazon S3
- Loading Data From Amazon S3
- COPY
- COPY Parameters
Viewing Profile Data in MC

Management Console allows you to view profile data about a single query. You can:

- Review the profile data in multiple views
- View details about projection metadata, execution events, and optimizer events
- Identify how much time was spent in each phase of query execution and which phases took the most amount of time

After you select the database you want to use, you can view the profile data using Management Console in either of two ways:

- Focus on specific areas of database activity, such as spikes in CPU usage
- Review the profile data for a specific query

To focus on specific areas of database activity:

1. At the bottom of the Management Console window, click the Activity tab.
2. From the list at the top of the page, select Queries.
3. On the activity graph, click the data point that corresponds to the query you want to view.
4. In the View Plan column, click Profile next to the command for which you want to view the query plan. Only certain queries, like SELECT, INSERT, UPDATE, and DELETE, have profile data.
5. In The Explain Plan window, Vertica profiles the query.
6. You can view the output in Path Information view, Query Plan Drilldown view, Tree Path view, or Profile Analysis view. To do so, click the respective buttons on the left of the output box.
To review the profile data for a specific query:

1. In the Explain window, type or paste the query text into the text box. Additionally, you can monitor queries that are currently running. To do so, perform one of the following. In the Find a Query By ID input window:

   - Enter the query statement and transaction ID
   - Click the Browse Running Queries link

   **Caution:** If you enter more than one query, Management Console profiles only the first query.

2. To receive periodic updates about the query's progress and resources used, select the Enable Monitoring check box. As a best practice, avoid specifying an interval time of less than 60 seconds because doing so may slow your query's progress.

3. Click the Profile button.

   While Vertica is profiling the query, a Cancel Query button is enabled briefly, allowing you to cancel the query and profiling task. If the Cancel Query button is disabled, that means Management Console does not have the proper information to cancel the query or the query is no longer running in the database.

   When processing completes, the profile data and metrics display below the text box. You can view the output in Path Information view, Query Plan Drilldown view, Tree Path view, or Profile Analysis view. To do so, click the respective view buttons on the left of the output box.

Viewing Different Profile Outputs

Vertica Management Console allows you to examine the results of your query profile in multiple views. You can view your profile in the following formats:

- Path Information view
- Query Drilldown view
- Tree Path view
- Profile Analysis view

You can change the query profile output using the icons on the bottom portion of the Explain page.

The Path Information view displays the query plan path along with metric data. If you enable profile monitoring, the data will update at the specified interval. To view metadata for a projection or a column, click the object name in the path output. A pop-up window displays the metadata if it is available.

The Query Plan Drilldown view shows detailed counter information at the node and operator level.
For each path, the path number is listed along with statistical information on the node and operator level. This view allows you to see which nodes are acting as outliers. Click on any of the bars to expand details for that node.

The Tree Path details the query plan in the form of a tree. If you enable monitoring, the state of the path blocks will change depending on whether the path is running, done, or has not started. Metric information is displayed in each path block for the counters you specified in the Profile Settings.

The Profile Analysis view allows you to identify any resource outliers. You can compare the estimated rows produced count with the actual rows produced count, view execution time per path, and identify memory usage per path.
When you profile a query, you will also see a pie chart detailing the query phase duration. You can also view projection metadata, execution events, and optimizer events by clicking on the respective buttons next to the pie chart.

Monitoring Profiling Progress

While loading profile data for a query, Management Console can provide updates about the query's progress and resources used.

To enable profiling progress updates, select the Enable Monitoring check box when profiling a query. See Viewing Profile Data in Management Console.

The default interval time is 60 seconds. At the specified interval, Management Console displays an updated view of the query's progress. Note that interval times of less than 60 seconds may slow down your query.
Viewing Updated Profile Metrics

At every interval, Management Console displays a new set of profile metrics. You can view these metrics in Path Information view, Query Plan Drilldown view, or Tree view by clicking the respective view buttons on the left of the output box.

- A dark blue bar indicates the current metric percentage.
- When a metric bar has decreased, a dark blue line indicates the previous metric percentage.
- When a metric bar has increased, a light blue bar indicates the added percentage. The previous percentage appears as a dark blue bar.
- A metric bar highlighted in yellow indicates it has changed since the last interval.
- A metric bar highlighted in red indicates the absolute value of the metric has decreased. This typically means Vertica reported the previous value incorrectly, and has readjusted. (For example, if Vertica previously reported path's Time value as 75 seconds, then reports it as 50 seconds at the next interval, the metric bar turns red to indicate the decrease in absolute Time value.)

Expanding and Collapsing Query Path Profile Data

When you have a query on the EXPLAIN window, the profile data displays in the right-hand side of the lower half of the window. The query path information can be lengthy, so you can collapse path information that is uninteresting, or expand paths that you want to focus on.
To collapse all the query paths, click Collapse All.

To expand all the query paths, click Expand All.

To expand an individual query path so you can see details about that step in processing the query, click the first line of the path information. Click the first line again to collapse the path data.

For information about what the profile data means, see About Profile Data in Management Console.

**About Profile Data in Management Console**

After you profile a specific query, the Management Console Explain page displays profile data like query duration, projection metadata, execution events, optimizer events, and metrics in a pie chart.

See the following links for more information on the kinds of profile data you can review on the Management Console Explain page:

- Projection Metadata
- Query Phase Duration
- Profile Metrics
- Execution Events
- Optimizer Events

**Projection Metadata**

To view projection metadata for a specific projection, click the projection name in the EXPLAIN output. Metadata for that projection opens in a pop-up window.

To view projection data for all projections accessed by that query, click the View Projection Metadata button at the top of the Explain page. The metadata for all projections opens in a new browser window.

**Note:** If the View Projection Metadata button is not enabled, click **Profile** to retrieve the profile data, including the projection metadata.
The projection metadata includes the following information:

- Projection ID
- Schema name
- Whether or not it is a superprojection
- Sort columns
- IDs of the nodes the projection is stored on
- Whether or not it is segmented
- Whether or not it is up to date
- Whether or not it has statistics
- Owner name
- Anchor table name

To display a SQL script that can recreate the projection on a different cluster, click Click to get export data. This script is identical to the output of the EXPORT_OBJECTS function. The SQL script opens in a pop-up window.

Copy and paste the command from this window, and click Close.

**Query Phase Duration**

This pie chart appears in the upper-right corner of the Query Plan window. It shows what percentage of total query processing was spent in each phase of processing the query.

The phases included in the pie chart (when applicable) are:

- Plan
- InitPlan
- SerializePlan
- PopulateVirtualProjection
- PreparePlan
- CompilePlan
- ExecutePlan
- AbandonPlan

Hover over the slices on the pie chart or over the names of the phases in the box to get additional information. You can see the approximate number of milliseconds (ms) and percentage used during each phase.

**Note:** The time data in the profile metrics might not match the times in the query phase duration. These times can differ because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Network latency can add more data, which is not taken into account in these calculations.

### Profile Metrics

In the Path Information view, the area to the right of each query path contains profile metrics for that path.

- **Disk**—Bytes of data accessed from disk by each query path. If none of the query paths accessed the disk data, all the values are 0.

- **Memory**—Bytes of data accessed from memory by each query path.

- **Sent**—Bytes of data sent across the cluster by each query path.

- **Received**—Bytes of data received across the cluster by each query path.

- **Time**—Number of milliseconds (ms) that the query path took to process on a given node, shown on progress bars. The sum of this data does not match the total time required to execute the query. This mismatch occurs because many tasks are executed in parallel on different nodes.

Hover over the progress bars to get more information, such as total bytes and percentages.

**Note:** The time data in the profile metrics might not match the times in the Query Phase Duration chart. These times can differ because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Network latency can add more data, which is not taken into account in these calculations.
## Execution Events

To help you monitor your database system, Vertica logs significant events that affect database performance and functionality. Click View Execution Events to see information about the events that took place while the query was executing.

If the View Execution Events button is not enabled, click Profile to retrieve the profile data, including the execution events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The execution events are described in the following table.

<table>
<thead>
<tr>
<th>Event Characteristic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Clock time when the event took place.</td>
</tr>
<tr>
<td>Node Name</td>
<td>Name of the node for which information is listed.</td>
</tr>
<tr>
<td>Session ID</td>
<td>Identifier of the session for which profile information is captured.</td>
</tr>
<tr>
<td>User ID</td>
<td>Identifier of the user who initiated the query.</td>
</tr>
<tr>
<td>Request ID</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Type of event processed by the execution engine. For a list of events and their descriptions, see Initial Process for Improving Query Performance.</td>
</tr>
<tr>
<td>Event Description</td>
<td>Generic description of the event.</td>
</tr>
<tr>
<td>Operator Name</td>
<td>Name of the Execution Engine component that generated the event. Examples include but are not limited to:</td>
</tr>
<tr>
<td></td>
<td>- DataSource</td>
</tr>
<tr>
<td></td>
<td>- DataTarget</td>
</tr>
<tr>
<td></td>
<td>- NetworkSend</td>
</tr>
<tr>
<td></td>
<td>- NetworkRecv</td>
</tr>
<tr>
<td></td>
<td>- StorageUnion</td>
</tr>
</tbody>
</table>
Values from the Operator name and Path ID columns let you tie a query event back to a particular operator in the query plan. If the event did not come from a specific operator, then this column is NULL.

Path ID | Unique identifier that Vertica assigns to a query operation or a path in a query plan. If the event did not come from a specific operator, this column is NULL.
--- | ---
Event OID | A unique ID that identifies the specific event.
Event Details | A brief description of the event and details pertinent to the specific situation.
Suggested Action | Recommended actions (if any) to improve query processing.

**Optimizer Events**

To help you monitor your database system, Vertica logs significant events that affect database performance and functionality. Click View Optimizer Events to see a table of the events that took place while the optimizer was planning the query.

If the View Optimizer Events button is not enabled, click Profile to retrieve the profile data, including the optimizer events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The following types of optimizer events may appear in the table:

<table>
<thead>
<tr>
<th>Event characteristic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Clock time when the event took place.</td>
</tr>
<tr>
<td>Node Name</td>
<td>Name of the node for which information is listed.</td>
</tr>
<tr>
<td>Session ID</td>
<td>Identifier of the session for which profile information is captured.</td>
</tr>
<tr>
<td>User ID</td>
<td>Identifier of the user who initiated the query.</td>
</tr>
<tr>
<td>Request ID</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Type of event processed by the optimizer.</td>
</tr>
</tbody>
</table>
### Event Description

- **Event Description**: Generic description of the event.
- **Event OID**: A unique ID that identifies the specific event.
- **Event Details**: A brief description of the event and details pertinent to the specific situation.
- **Suggested Action**: Recommended actions (if any) to improve query processing.

---

## Clearing Query Data

After you finish reviewing the current query data, click Clear All to clear the query text and data. Alternatively, to display information about another query, enter the query text and click Explain or Profile.
Extended Monitoring

Enabling extended monitoring allows you to monitor a longer range of data through MC. This can offer insight into long-term trends in your database's health. MC can also continue to display your monitored database's dashboard while it is down.

Extended monitoring uses Kafka to stream monitoring data from your monitored databases to a single MC storage database. MC can query the storage database instead of your monitored database to render some of its charts, reducing impact on your monitored database's performance.

How Extended Monitoring Works

By default, MC monitors your database by querying it directly for monitoring data about system activities, performance, and resource utilization. Typically, the Data Collector stores all monitoring data in data collector (DC) tables. However, DC tables have limited retention periods. See Retaining Monitoring Information.

Extended monitoring stores your database's monitoring data in a dedicated storage database. Vertica streams data from your database's DC tables through Kafka servers to the storage database. To use extended monitoring, you must have access to a running Kafka server. For more how Vertica integrates with Kafka, see Integrating with Apache Kafka.

After you set up and enable extended monitoring for a monitored database, MC renders several of your database's charts and graphs by querying the MC storage database instead of directly querying the database you are monitoring.

You can enable extended monitoring for any, or all, of your monitored databases. The MC storage database provides a single repository for monitoring data from every database that uses enabled extended monitoring.

In the following example, Kafka streams system data from two monitored databases to the storage database. MC uses the storage database to render individual dashboards for each monitored database. Be aware that MC always creates a dashboard that monitors the MC storage database.
Use Extended Monitoring

Important: To use extended monitoring, OpenText recommends installing Management Console on a host without any other Vertica databases.

When a database has extended monitoring enabled, the MC charts that use the feature display a rocket ship icon in the corner. You can use these charts to access longer-term data about your database's health or performance.

To view historical information in these charts, click the calendar icon to specify the timeframe to display. For example, if your database has been down for several hours, your charts do not display recent activity in your database. You could use the timeframe filter in the System Bottlenecks chart to see unusual resource usage occurred in your database in the hour it went down.

You can view a history of the Kafka streaming jobs loading data into the storage database. MC displays these jobs on the Load tab of your storage database's dashboard. See Viewing Load History.

Set Up Extended Monitoring

To set up extended monitoring, see Managing the Storage Database and Managing Extended Monitoring on a Database.
See Also

- Managing the Storage Database
- Managing Extended Monitoring on a Database
- Viewing Load History
- Integrating with Apache Kafka
- Retaining Monitoring Information

Managing the Storage Database

Extended Monitoring stores your Vertica database's monitoring data in a dedicated MC storage database.

To use Extended Monitoring, you must first set up the storage database and configure it for Kafka streaming. Then, turn on Extended Monitoring for any or all monitored databases.

MC automatically configures a schema for the storage database, named dcschema, which is synced with DC tables on your monitored databases.

Caution: Do not alter dcschema after MC has configured it. Altering it could cause the storage database to lose data or supply incorrect monitoring information to MC.

MC Preparation

First verify that MC is not installed on the same host as a Vertica database. When Extended Monitoring is enabled, MC sharing a host with a production database can affect performance.

You must also increase the allocation of memory for the MC application server. See Modify Memory Allocation. Tune these options based on:

- The demands of your database
- The amount of monitoring data you plan to view in MC charts at once.

For example, MC requires more memory to display a week of data in a chart.
Modify Memory Allocation

To modify memory allocation:

1. In Management Console, select the Configuration tab on the MC Settings page.

2. Modify the following fields under Application Server JVM Settings to increase the allocation of memory for the JVM:
   - **Initial Heap Size:** For Extended Monitoring, a minimum value of 2GB is recommended. (The default value is 1GB.)
   - **Maximum Heap Size:** For Extended Monitoring, a minimum value of 4GB is recommended. (The default value is 2GB.)

3. Click Apply at the top right of the page. A prompt appears to restart MC.

4. Click OK to restart MC and save your changes.

Storage Database Requirements

To set up storage for Extended Monitoring, your system must meet the following prerequisites:

- Vertica 8.x

- An available host, or available database whose Vertica version corresponds to the version of the database you plan to monitor. (For example, to use Extended Monitoring for a Vertica 8.1.0 database, its storage database should also be on Vertica 8.1.0.)

- Configured MC for Extended Monitoring (See MC Preparation)

- Access to a deployed Kafka server (For details on installing Kafka, see the Apache Kafka site)

Set Up the Storage Database

To set up the storage database for Extended Monitoring:
1. On the MC Settings page, select the MC Storage DB Setup tab.

2. In the Server field in the Kafka Broker section, enter the host name or IP addresses and ports for one or more of your deployed Kafka servers.

3. Designate the storage database in the MC External Storage Database section. You can create a new database, or use an existing database.
   - Create a new database: Choose this option to create a new single node cluster on an available host using a Community Edition license of Vertica. This choice does not affect your normal Vertica license usage.
   - Use an existing database known to MC: Choose this option to designate a database you have already imported to MC. If the schema 'dcschema' exists in the database, a dialog appears:
     - Click Append to keep the existing schema's data. For example, if you have already used this database for Extended Monitoring storage and are re-importing it, you can use this option to retain its historical data for continued use.
     - Click Remove to clear the existing schema from the database, and creates a fresh version of dcschema configured for Extended Monitoring storage.

4. Click Enable Streaming to enable data streaming from the Kafka server to the MC storage database.

5. Turn on Extended Monitoring for your databases on the Extended Monitoring tab. See Managing Extended Monitoring on a Database for more information.

Restart the Storage Database

If you stop the storage database while streaming is enabled, streaming to the storage database stops automatically. You must re-enable streaming on the MC Storage DB Setup tab after you restart the storage database.

If streaming to the MC storage database is disabled while Extended Monitoring on your database is on, the Kafka retention policy determines how long streaming can remain disabled without data loss. See Managing Streaming Services for Extended Monitoring.
Discontinue the Storage Database

1. Select the Extended Monitoring tab in MC Settings.
2. Set Extended Monitoring for all databases to OFF.
3. Select the MC Storage DB Setup tab in MC Settings.
4. Click Disable Streaming in the MC External Storage Database section to de-activate your storage database.
5. Click Remove in the MC External Storage Database section to remove the MC Storage Database from MC.
6. Choose whether to keep or remove the data your storage database has collected:
   - Keep Data: Existing data will not removed. If you re-use this database for Extended Monitoring storage, you can choose to append new collected monitoring data to this existing data.
   - Remove Data: MC deletes its customized storage schema from the database.

Configure Storage Database Memory Usage

On the Resource Pools tab of the storage database, you can optionally increase the memory size of SYSQUERY and KAFKA_DEFAULT_POOL. For setting resource pool parameters in MC, see Configuring Resource Pools in Management Console.

- **SYSQUERY**: Reserved for temporary storage of intermediate results of queries against system monitoring and catalog tables. Default setting is 64M. For best performance for MC, set to 2G or higher.

- **KAFKA_DEFAULT_POOL**: Reserved for all queries executed by the Kafka scheduler. Default setting is 30%, which is the recommended setting. By default, queries spill to the general pool when they exceed the 30% memory size.

Manage Disk Space

The storage database uses a customized schema, named dcschema. You can monitor these tables on MC, using the Table Utilization chart on the storage database’s Activity tab. The Table
Utilization chart lists all the tables in dcschema and their details, such as row counts and column properties. You can sort by row count to determine if certain tables use more disk space on your storage database. See Monitoring Table Utilization and Projections.

You should regularly drop partitions from dcschema if you have limited disk space for the MC storage database. MC does not automatically drop partitions from the storage database. For more information on dropping partitions, see Dropping Partitions.

The table dc_execution_engine_profiles is partitioned by day. Because this table typically contains the most rows, as a best practice you should drop partitions from this table more often. The following example shows how you can specify partition key 2016-08-22 to drop a partition from dc_execution_engine_profiles.

```
SELECT DROP_PARTITIONS
  ('dcschema.dc_execution_engine_profiles', 2016-08-2, 2016-08-22);
```

Other than dc_execution_engine_profiles, all other tables in dcschema are partitioned by week. The next example shows you how you can drop a partition from the table dc_cpu_aggregate_by_minute, specifying the thirty-fourth week of 2016.

```
SELECT DROP_PARTITION
  ('dcschema.dc_cpu_aggregate_by_minute', 201634, 201634);
```

Manage Client Sessions

By default Vertica allows 50 client sessions and an additional five administrator sessions per node. If you reach the limit on the storage database, MC switches back to default monitoring, and does not use Extended Monitoring data from the storage database.

You can optionally configure the maximum number of client sessions that can run on a single database cluster node on your MC storage database’s Settings page:

1. On the storage database dashboard, click the Settings page.
2. Choose the General tab.
3. Input a value in the Maximum client sessions field. Valid values for the parameter are 0 to 1000.

For more details about managing client connections in MC, see Managing Client Connections.
See Also

- Extended Monitoring
- Managing Extended Monitoring on a Database
- Viewing Load History

Managing Extended Monitoring on a Database

When you enable extended monitoring on your Vertica database, monitoring data from your database streams through Kafka servers to the MC storage database.

You can enable streaming for any or all databases that MC monitors.

Extended Monitoring Prerequisites

Before you can enable extended monitoring, your system must meet these prerequisites:

- Vertica 8.x
- Deployed Kafka server(s)
- Configured MC for extended monitoring (See Managing the Storage Database)
- Deployed MC storage database (See Managing the Storage Database)

Enable Extended Monitoring

1. Select the Extended Monitoring tab on MC Settings.
   
   The Extended Monitoring page displays all databases monitored by MC.

2. In the Memory Limit field for the database of your choice, set the maximum amount of memory the database can use for streaming monitoring data. For more about the memory limit, see Managing Streaming Services for Extended Monitoring.
3. In the Extended Monitoring column, select ON to enable streaming for the database of your choice.

   The database begins streaming its monitoring data to the Kafka server.

User Access

When you change user permissions for a database using extended monitoring, the user access policy on the storage database does not automatically update. On the Extended Monitoring page, in the user access column for your database, click Refresh to sync the policy.

If you rename a Vertica user, you must re-map the user in MC Settings before refreshing the user access policy.

See Also

- Extended Monitoring
- Managing the Storage Database
- Viewing Load History
- Integrating with Apache Kafka

Managing Streaming Services for Extended Monitoring

When extended monitoring is enabled, Vertica streams data from your database through Kafka servers to the storage database.

For additional parameters that optimize the performance of Kafka with Vertica, see Kafka and Vertica Configuration Settings.
View Streaming Details in MC

Click the Load tab on your database's MC dashboard to see the Data Load Activity page. On this page, the Continuous tab displays details about all continuous loading jobs for extended monitoring. You can use this page to monitor whether your extended monitoring data is streaming successfully to the MC storage database.

See Viewing Load History for more about the Data Load Activity page.

Tip: If you do not see loading jobs for extended monitoring, verify that you have selected Show MC data collector monitoring streams at the top of the Continuous tab.

Prevent Data Loss

The Memory Limit buffer allows you to restart the Kafka server without data loss. Vertica queues the streamed data until you restart the Kafka server. When the Kafka server remains down for an extended period of time, data loss occurs when the queue of streamed data exceeds the buffer. You set the buffer size on the Extended Monitoring tab when you enable extended monitoring for a database. See Managing Extended Monitoring on a Database.

The Kafka retention policy determines when data loss occurs during the following scenarios:

- Restarting the MC storage database (see Managing the Storage Database)
- Disabling streaming on the MC storage database (see Managing the Storage Database)
- Restart a micro-batch (see Loading Data Using MC)

The Kafka retention policy can allow you to restart these extended monitoring components without data loss. The Kafka server retains the data while the listed components are disabled. Data loss occurs when the streamed data exceeds the Kafka retention policy's log size or retention time limits. See the Apache Kafka documentation for how to configure the retention policy.

Changing the Kafka Server

Be aware that when you change Kafka servers for extended monitoring on the MC Storage DB Setup page, you must disable all extended monitoring processes and re-configure the
MC storage database. For storage database setup instructions, see Managing the Storage Database.

See Also

- Managing Extended Monitoring on a Database
- Viewing Load History
- Kafka and Vertica Configuration Settings
- Integrating with Apache Kafka
Welcome to the Vertica SQL Reference Manual. This guide provides an overview of Vertica Structured Query Language (SQL). It defines system limits, describes SQL Language elements and system tables. This guide also provides reference descriptions of SQL Data Types, SQL Functions, and SQL Statements.

This document assumes that you are familiar with the basic concepts and terminology of the SQL language and relational database management systems.
**SQL in Vertica**

Vertica offers a robust set of SQL elements that allow you to manage and analyze massive volumes of data quickly and reliably. Vertica uses the following:

**SQL language elements**, including:

- Keywords and Reserved Words
- Identifiers
- Literals
- Operators
- Expressions
- Predicates
- Hints

**SQL data types**, including:

- Binary
- Boolean
- Character
- Date/Time
- Long
- Numeric

**SQL functions** including Vertica-specific functions that take advantage of Vertica's unique column-store architecture. For example, call `ANALYZE_STATISTICS` to collect and aggregate a variable amount of sample data for statistical analysis.

**SQL statements** that let you write robust queries to quickly return large volumes of data.
# System Limits

This section describes the system limits on the size and number of objects in a Vertica database. In most cases, computer memory and disk drive are the limiting factors.

<table>
<thead>
<tr>
<th>Item</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Maximum 128 (without Vertica assistance)</td>
</tr>
<tr>
<td>Database size</td>
<td>Approximates the number of files times the file size on a platform, depending on the maximum disk configuration</td>
</tr>
<tr>
<td>Table size</td>
<td>The smaller of:</td>
</tr>
<tr>
<td></td>
<td>• $2^{64}$ rows per node</td>
</tr>
<tr>
<td></td>
<td>• $2^{63}$ bytes per column</td>
</tr>
<tr>
<td>Row size</td>
<td>32,768,000 bytes</td>
</tr>
<tr>
<td></td>
<td>Row size is approximately the sum of its maximum column sizes, where, for example, a VARCHAR(80) has a maximum size of 80 bytes.</td>
</tr>
<tr>
<td>Key size</td>
<td>Limited only by row size</td>
</tr>
<tr>
<td>Tables/projections per database</td>
<td>Limited by physical RAM, as the catalog must fit in memory.</td>
</tr>
<tr>
<td>Concurrent connections per node</td>
<td>Default of 50, limited by physical RAM (or threads per process), typically 1024</td>
</tr>
<tr>
<td>Concurrent connections per cluster</td>
<td>Limited by physical RAM of a single node (or threads per process), typically 1024</td>
</tr>
<tr>
<td>Columns per table</td>
<td>1600</td>
</tr>
<tr>
<td>Rows per load</td>
<td>$2^{63}$</td>
</tr>
<tr>
<td>ROS containers per projection</td>
<td>1024</td>
</tr>
<tr>
<td>Length of fixed-length column</td>
<td>65000 bytes</td>
</tr>
<tr>
<td>Item</td>
<td>Limit</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Length of variable-length column</td>
<td>32000000 bytes</td>
</tr>
<tr>
<td>Length of basic names</td>
<td>128 bytes. Basic names include table names, column names, etc.</td>
</tr>
<tr>
<td>Query length</td>
<td>No limit</td>
</tr>
<tr>
<td>Depth of nesting subqueries</td>
<td>Unlimited in FROM, WHERE, and HAVING clauses</td>
</tr>
</tbody>
</table>
SQL Language Elements

This chapter presents detailed descriptions of the language elements and conventions of Vertica SQL.

Keywords

Keywords are words that have a specific meaning in the SQL language. Every SQL statement contains one or more keywords. Although SQL is not case-sensitive with respect to keywords, they are generally shown in uppercase letters throughout this documentation for readability purposes.

Note: If you use a keyword as the name of an identifier or an alias in your SQL statements, you may have to qualify the keyword with AS or double-quotes. Vertica requires AS or double-quotes for certain reserved and non-reserved words to prevent confusion with expression syntax, or where the use of a word would be ambiguous.

Reserved Words and Keywords

Many keywords are also reserved words.

Vertica recommends that you not use reserved words as names for objects, or as identifiers. Including reserved words can make your SQL statements confusing. Where you need to use reserved words as names for objects or as identifiers, you need to double-quote them. An example of where you might need to do this is where Vertica has added a reserved word that you already use throughout your existing SQL scripts.

Note: All reserved words are also keywords, but Vertica can add reserved words that are not keywords. A reserved word can simply be a word that is reserved for future use.

Non-reserved Keywords

Non-reserved keywords have a special meaning in some contexts, but can be used as identifiers in others. You can use non-reserved keywords as aliases—for example, SOURCE:

```sql
=> SELECT my_node AS SOURCE FROM nodes;
```
**Note:** Vertica uses several non-reserved keywords in directed queries to specify special join types. You can use these keywords as table aliases only if they are double-quoted; otherwise, double-quotes can be omitted:

- ANTI
- NULLAWARE
- SEMI
- SEMIALL
- UNI

**Viewing the List of Reserved and Non-reserved Keywords**

To view the current list of Vertica reserved and non-reserved words, query the `KEYWORDS` system table.

```
VMart=> select * from keywords;
```

The output lists keywords alphabetically and identifies them as reserved (R) or non-reserved (N).

**Identifiers**

Identifiers (names) of objects such as schema, table, projection, column names, and so on, can be up to 128 bytes in length.

**Unquoted Identifiers**

Unquoted SQL identifiers must begin with one of the following:

- Letters (use A–Z or a-z only; behavior for Unicode is undefined)
- Underscore (_)

Subsequent characters in an identifier can be:
• Letters (as above)

• Unicode letters (letters with diacriticals or not in the Latin alphabet)

• Underscore (\_)

• Digits (0–9)

• Dollar sign ($)

Caution: The SQL standard does not support dollar sign in identifiers so usage can cause application portability problems.

Important: Models do not support unicode letters or the dollar sign.

Quoted Identifiers

Identifiers enclosed in double quote (\”) characters can contain any character. If you want to include a double quote, you need a pair of them; for example """". You can use names that would otherwise be invalid, such as names that include only numeric characters ("123") or contain space characters, punctuation marks, keywords, and so on; for example:

CREATE SEQUENCE "my sequence!";

Double quotes are required for non-alphanumerics and SQL keywords such as "1time", "Next week" and "Select".

Case Sensitivity

Identifiers are not case-sensitive. Thus, identifiers "ABC", "ABc", and "aBc" are synonymous, as are ABC, ABc, and aBc.

Non-ASCII Characters

Vertica accepts non-ASCII UTF-8 Unicode characters for table names, column names, and other Identifiers, extending the cases in which upper/lower case distinctions are ignored (case-folded) to all alphabets, including Latin, Cyrillic, and Greek.

For example, the following CREATE TABLE statement uses the ß (German eszett) in the table name:
CREATE TABLE straße(x int, y int);

Identifiers Are Stored As Created

SQL identifiers, such as table and column names, are no longer converted to lowercase. They are stored as created, and references to them are resolved using case-insensitive compares. It is not necessary to double quote mixed-case identifiers. For example, the following statement creates table ALLCAPS.

```sql
=> CREATE TABLE ALLCAPS(c1 varchar(30));
=> INSERT INTO ALLCAPS values('upper case');
```

The following statements are variations of the same query and all return identical results:

```sql
=> SELECT * FROM ALLCAPS;
=> SELECT * FROM allcaps;
=> SELECT * FROM "allcaps";
```

All three commands return the same result:

```
c1
----------
upper case
(1 row)
```

Note that the system returns an error if you try to create table AllCaps:

```sql
=> CREATE TABLE AllCaps(c1 varchar(30));
ROLLBACK: table "AllCaps" already exists
```

See QUOTE_IDENT for additional information.
**Literals**

Literals are numbers or strings used in SQL as constants. Literals are included in the select-list, along with expressions and built-in functions and can also be constants.

Vertica provides support for number-type literals (integers and numerics), string literals, VARBINARY string literals, and date/time literals. The various string literal formats are discussed in this section.

**Number-Type Literals**

Vertica supports three types of numbers: integers, numerics, and floats.

- **Integers** are whole numbers less than 2^63 and must be digits.
- **Numerics** are whole numbers larger than 2^63 or that include a decimal point with a precision and a scale. Numerics can contain exponents. Numbers that begin with 0x are hexadecimal numerics.

Numeric-type values can also be generated using casts from character strings. This is a more general syntax. See the Examples section below, as well as Data Type Coercion Operators (CAST).

**Syntax**

```
digits
digits.[digits] | [digits].digits
digits e[+-]digits | [digits].digits e[+-]digits | digits.[digits] e[+-]digits
```

**Parameters**

<table>
<thead>
<tr>
<th>digits</th>
<th>One or more numeric characters, 0 through 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Exponent marker</td>
</tr>
</tbody>
</table>
Notes

- At least one digit must follow the exponent marker (e), if e is present.
- There cannot be any spaces or other characters embedded in the constant.
- Leading plus (+) or minus (–) signs are not considered part of the constant; they are unary operators applied to the constant.
- In most cases a numeric-type constant is automatically coerced to the most appropriate type depending on context. When necessary, you can force a numeric value to be interpreted as a specific data type by casting it as described in Data Type Coercion Operators (CAST).
- Floating point literals are not supported. If you specifically need to specify a float, you can cast as described in Data Type Coercion Operators (CAST).
- Vertica follows the IEEE specification for floating point, including NaN (not a number) and Infinity (Inf).
- A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.
- Dividing INTEGERS (x / y) yields a NUMERIC result. You can use the // operator to truncate the result to a whole number.

Examples

The following are examples of number-type literals:

```
42
3.5
4.
.001
5e2
1.925e-3
```

Scientific notation:

```
=> SELECT NUMERIC '1e10';
?column?  
----------
10000000000  (1 row)
```
**BINNARY scaling:**

```sql
=> SELECT NUMERIC '1p10';
?column?
---------
1024
(1 row)
=> SELECT FLOAT 'Infinity';
?column?
---------
Infinity
(1 row)
```

The following examples illustrated using the `/` and `//` operators to divide integers:

```sql
=> SELECT 40/25;
?column?
-----------------
1.600000000000000000
(1 row)
=> SELECT 40//25;
?column?
---------
1
(1 row)
```

**See Also**

- Data Type Coercion
String Literals

String literals are string values surrounded by single or double quotes. Double-quoted strings are subject to the backslash, but single-quoted strings do not require a backslash, except for \' and \\.

You can embed single quotes and backslashes into single-quoted strings.

To include other backslash (escape) sequences, such as \t (tab), you must use the double-quoted form.

Precede single-quoted strings with a space between the string and its preceding word, since single quotes are allowed in identifiers.

See Also

- **SET STANDARD_CONFORMING_STRINGS**
- **SET ESCAPE_STRING_WARNING**
- Internationalization Parameters
- Implement Locales for International Data Sets

Character String Literals

Character string literals are a sequence of characters from a predefined character set and are enclosed by single quotes. If the single quote is part of the sequence, it must be doubled as "'".

Syntax

`'characters'`

Parameters

| characters | Arbitrary sequence of characters bounded by single quotes ('') |
Single Quotes in a String

The SQL standard way of writing a single-quote character within a string literal is to write two adjacent single quotes. For example:

```sql
=> SELECT 'Chester''s gorilla';
?column?
------------------
Chester's gorilla
(1 row)
```

Standard Conforming Strings and Escape Characters

Vertica uses standard conforming strings as specified in the SQL standard, which means that backslashes are treated as string literals, not escape characters.

**Note:** Earlier versions of Vertica did not use standard conforming strings, and backslashes were always considered escape sequences. To revert to this older behavior, set the `StandardConformingStrings` parameter to '0', as described in Configuration Parameters in the Administrator's Guide.

Examples

```sql
=> SELECT 'This is a string';
?column?
------------------
This is a string
(1 row)
=> SELECT 'This \ is a string';
  WARNING: nonstandard use of escape in a string literal at character 8
  HINT: Use the escape string syntax for escapes, e.g., E'\r\n'.
?column?
------------------
This is a string
(1 row)
vmartdb=> SELECT E'This \is a string';
?column?
------------------
This is a string
(1 row)
=> SELECT E'This is a \new line';
?column?
------------------
This is a
new line
(1 row)
=> SELECT 'String''s characters';
?column?
------------------
------------------
```
See Also

- SET STANDARD_CONFORMING_STRINGS
- SET ESCAPE_STRING_WARNING
- Internationalization Parameters
- Implement Locales for International Data Sets

Dollar-Quoted String Literals

Dollar-quoted string literals are rarely used, but are provided here for your convenience.

The standard syntax for specifying string literals can be difficult to understand. To allow more readable queries in such situations, Vertica SQL provides dollar quoting. Dollar quoting is not part of the SQL standard, but it is often a more convenient way to write complicated string literals than the standard-compliant single quote syntax.

Syntax

$$characters$$

Parameters

| characters | Arbitrary sequence of characters bounded by paired dollar signs ($$) |

Dollar-quoted string content is treated as a literal. Single quote, backslash, and dollar sign characters have no special meaning within a dollar-quoted string.

Notes

A dollar-quoted string that follows a keyword or identifier must be separated from the preceding word by whitespace; otherwise, the dollar-quoting delimiter is taken as part of the preceding identifier.
Examples

```sql
=> SELECT $$Fred's\n car$$;
 ?column?
 ---------------------
 Fred's\n car
 (1 row)

=> SELECT 'SELECT 'fact'';
 ERROR: syntax error at or near ""';" at character 21
 LINE 1: SELECT 'SELECT 'fact'';

=> SELECT $$'fact''$$;
 ?column?
 ---------------------
 SELECT $$fact$$
 (1 row)

=> SELECT '"fact'''';
 ?column?
 ---------------------
 SELECT 'fact';
 (1 row)
```

Unicode String Literals

Syntax

```
U&'characters' [ UESCAPE 'Unidentified escape character' ]
```

Parameters

<table>
<thead>
<tr>
<th>characters</th>
<th>Arbitrary sequence of UTF-8 characters bounded by single quotes ('')</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicode escape character</td>
<td>A single character from the source language character set other than a hexit, plus sign (+), quote ('), double quote ('&quot;), or white space</td>
</tr>
</tbody>
</table>

Using Standard Conforming Strings

With StandardConformingStrings enabled, Vertica supports SQL standard Unicode character string literals (the character set is UTF-8 only).
Before you enter a Unicode character string literal, enable standard conforming strings in one of the following ways.

- To enable for all sessions, update the StandardConformingStrings configuration parameter. See Configuration Parameters in the Administrator's Guide.

- To treat backslashes as escape characters for the current session, use the SET STANDARD_CONFORMING_STRINGS statement.

See also Extended String Literals.

Examples

To enter a Unicode character in hexadecimal, such as the Russian phrase for "thank you, use the following syntax:

```
=> SET STANDARD_CONFORMING_STRINGS TO ON;
=> SELECT U'\0441\043F\0430\0441\0438\0431\043E' as 'thank you';
    thank you
-------------
  спасибо
(1 row)
```

To enter the German word müde (where u is really u-umlaut) in hexadecimal:

```
=> SELECT U'm\00fcde';
 ?column?
---------
 müde
(1 row)
=> SELECT 'ü';
 ?column?
---------
 ü
(1 row)
```

To enter the LINEAR B IDEOGRAM B240 WHEELED CHARiot in hexadecimal:

```
=> SELECT E'\xF0\x90\x83\x8C';
 ?column?
---------
(wheeled chariot character)
(1 row)
```

Note: Not all fonts support the wheeled chariot character.
See Also

- SET STANDARD_CONFORMING_STRINGS
- SET ESCAPE_STRING_WARNING
- Internationalization Parameters
- Implement Locales for International Data Sets

**VARBINARY String Literals**

VARBINARY string literals allow you to specify hexadecimal or binary digits in a string literal.

**Syntax**

\[
X'\text{<hexadecimal digits>}'
B'\text{<binary digits>}'
\]

**Parameters**

<table>
<thead>
<tr>
<th>X or x</th>
<th>Specifies hexadecimal digits. The <code>&lt;hexadecimal digits&gt;</code> string must be enclosed in single quotes (').</th>
</tr>
</thead>
<tbody>
<tr>
<td>B or b</td>
<td>Specifies binary digits. The <code>&lt;binary digits&gt;</code> string must be enclosed in single quotes (').</td>
</tr>
</tbody>
</table>

**Examples**

```
=> SELECT X'abcd';
?column?
---------
\253\315
(1 row)

=> SELECT B'101100';
?column?
---------
,          
(1 row)
```
Extended String Literals

Syntax

\E characters'\n
Parameters

<table>
<thead>
<tr>
<th>characters</th>
<th>Arbitrary sequence of characters bounded by single quotes ('')</th>
</tr>
</thead>
</table>

You can use C-style backslash sequence in extended string literals, which are an extension to the SQL standard. You specify an extended string literal by writing the letter E as a prefix (before the opening single quote); for example:

\E 'extended character string\n'

Within an extended string, the backslash character (\) starts a C-style backslash sequence, in which the combination of backslash and following character or numbers represent a special byte value, as shown in the following list. Any other character following a backslash is taken literally; for example, to include a backslash character, write two backslashes (\\).

- \ is a backslash
- \b is a backspace
- \f is a form feed
- \n is a newline
- \r is a carriage return
- \t is a tab
- \x##, where ## is a 1 or 2-digit hexadecimal number; for example \x07 is a tab
- \##, where ### is a 1, 2, or 3-digit octal number representing a byte with the corresponding code.

When an extended string literal is concatenated across lines, write only E before the first opening quote:
Two adjacent single quotes are used as one single quote:

```sql
=> SELECT 'Aren''t string literals fun?';
   ?column?
------------------------
 Aren't string literals fun?
   (1 row)
```

Standard Conforming Strings and Escape Characters

When interpreting commands, such as those entered in vsql or in queries passed via JDBC or ODBC, Vertica uses standard conforming strings as specified in the SQL standard. In standard conforming strings, backslashes are treated as string literals (ordinary characters), not escape characters.

**Note:** Text read in from files or streams (such as the data inserted using the `COPY` statement) are not treated as literal strings. The `COPY` command defines its own escape characters for the data it reads. See the `COPY` statement documentation for details.

The following options are available, but Vertica recommends that you migrate your application to use standard conforming strings at your earliest convenience, after warnings have been addressed.

- To revert to pre 4.0 behavior, set the `StandardConformingStrings` parameter to '0', as described in [Configuration Parameters](#) in the Administrator's Guide.

- To enable standard conforming strings permanently, set the `StandardConformingStrings` parameter to '1', as described in the procedure in the section, "Identifying Strings that are not Standard Conforming," below.

- To enable standard conforming strings per session, use `SET STANDARD_CONFORMING_STRING TO ON`, which treats backslashes as escape characters for the current session only.

The two sections that follow help you identify issues between Vertica 3.5 and 4.0.
Identifying Strings That Are Not Standard Conforming

The following procedure can be used to identify nonstandard conforming strings in your application so that you can convert them into standard conforming strings:

1. Be sure the StandardConformingStrings parameter is off, as described in Internationalization Parameters in the Administrator's Guide.

   ```sql
   => ALTER DATABASE mydb SET StandardConformingStrings = 0;
   ```

   **Note:** Vertica recommends that you migrate your application to use Standard Conforming Strings at your earliest convenience.

2. Turn on the EscapeStringWarning parameter. (ON is the default in Vertica Version 4.0 and later.)

   ```sql
   => ALTER DATABASE mydb SET EscapeStringWarning = 1;
   ```

   Vertica now returns a warning each time it encounters an escape string within a string literal. For example, Vertica interprets the `\n` in the following example as a new line:

   ```sql
   => SELECT 'a\nb';
   WARNING: nonstandard use of escape in a string literal at character 8
   HINT: Use the escape string syntax for escapes, e.g., E'\r\n'.
   ?column?
   a
   b
   (1 row)
   ```

   When StandardConformingStrings is ON, the string is interpreted as four characters: `a \ n b`.

   Modify each string that Vertica flags by extending it as in the following example:

   ```sql
   E'a\nb'
   ```

   Or if the string has quoted single quotes, double them; for example, `'one' double`.

3. Turn on the StandardConformingStrings parameter for all sessions:

   ```sql
   => ALTER DATABASE mydb SET StandardConformingStrings = 1;
   ```
Doubled Single Quotes

This section discusses vsql inputs that are not passed on to the server.

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, the following inputs, 'You''re here!' ignored the second consecutive quote and returns the following:

```sql
=> SELECT 'You''re here';
  ?column?
--------------
  You're here!
(1 row)
```

This is the SQL standard representation and is preferred over the form, 'You\'re here!', because backslashes are not parsed as before. You need to escape the backslash:

```sql
=> SELECT ('You\'re here');
  ?column?
--------------
  You're here!
(1 row)
```

This behavior change introduces a potential incompatibility in the use of the vsql \set command, which automatically concatenates its arguments. For example, the following works in both Vertica 3.5 and 4.0:

```bash
\set file \''pwd\'/file.txt' \''\echo :file
```

vsq| takes the four arguments and outputs the following:

```
'/home/vertica/file.txt'
```

In Vertica 3.5 the above \set file command could be written all with the arguments run together, but in 4.0 the adjacent single quotes are now parsed differently:

```bash
\set file '\''pwd\'/file.txt''\''\echo :file
 '/home/vertica/file.txt''
```

Note the extra single quote at the end. This is due to the pair of adjacent single quotes together with the backslash-quoted single quote.

The extra quote can be resolved either as in the first example above, or by combining the literals as follows:

```bash
\set file '\''pwd\'/file.txt''\''\echo :file
 '/home/vertica/file.txt''
```
In either case the backslash-quoted single quotes should be changed to doubled single quotes as follows:

```
\set file ''' `pwd` '/file.txt'''
```

Additional Examples

```
=> SELECT 'This \"is a string\"';
   ?column?
 ------------
 This \"is a string
 (1 row)
=> SELECT E'This \"is a string\"';
   ?column?
 ------------
 This is a string
=> SELECT E'This is a \n new line';
   ?column?
 ------------
 This is a
 new line
 (1 row)
=> SELECT 'String''s characters';
   ?column?
 ------------
 String's characters
 (1 row)
```
Date/Time Literals

Date or time literal input must be enclosed in single quotes. Input is accepted in almost any reasonable format, including ISO 8601, SQL-compatible, traditional POSTGRES, and others.

Vertica handles date/time input more flexibly than the SQL standard requires. The exact parsing rules of date/time input and for the recognized text fields including months, days of the week, and time zones are described in Date/Time Expressions.

Time Zone Values

Vertica attempts to be compatible with the SQL standard definitions for time zones. However, the SQL standard has an odd mix of date and time types and capabilities. Obvious problems are:

- Although the DATE type does not have an associated time zone, the TIME/TIMETZ type can. Time zones in the real world have little meaning unless associated with a date as well as a time, since the offset can vary through the year with daylight-saving time boundaries.

- Vertica assumes your local time zone for any data type containing only date or time.

- The default time zone is specified as a constant numeric offset from UTC. It is therefore not possible to adapt to daylight-saving time when doing date/time arithmetic across DST boundaries.

To address these difficulties, Vertica recommendations using Date/Time types that contain both date and time when you use time zones. Vertica recommends that you do not use the type TIME WITH TIME ZONE, even though it is supported for legacy applications and for compliance with the SQL standard.

Time zones and time-zone conventions are influenced by political decisions, not just earth geometry. Time zones around the world became somewhat standardized during the 1900's, but continue to be prone to arbitrary changes, particularly with respect to daylight-savings rules.

Vertica currently supports daylight-savings rules over the time period 1902 through 2038, corresponding to the full range of conventional UNIX system time. Times outside that range are taken to be in "standard time" for the selected time zone, no matter what part of the year in which they occur.
<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PST</td>
<td>Pacific Standard Time</td>
</tr>
<tr>
<td>-8:00</td>
<td>ISO-8601 offset for PST</td>
</tr>
<tr>
<td>-800</td>
<td>ISO-8601 offset for PST</td>
</tr>
<tr>
<td>-8</td>
<td>ISO-8601 offset for PST</td>
</tr>
<tr>
<td>zulu</td>
<td>Military abbreviation for UTC</td>
</tr>
<tr>
<td>z</td>
<td>Short form of zulu</td>
</tr>
</tbody>
</table>

**Day of the Week Names**

The following tokens are recognized as names of days of the week:

<table>
<thead>
<tr>
<th>Day</th>
<th>Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUNDAY</td>
<td>SUN</td>
</tr>
<tr>
<td>MONDAY</td>
<td>MON</td>
</tr>
<tr>
<td>TUESDAY</td>
<td>TUE, TUES</td>
</tr>
<tr>
<td>WEDNESDAY</td>
<td>WED, WEDS</td>
</tr>
<tr>
<td>THURSDAY</td>
<td>THU, THUR, THURS</td>
</tr>
<tr>
<td>FRIDAY</td>
<td>FRI</td>
</tr>
<tr>
<td>SATURDAY</td>
<td>SAT</td>
</tr>
</tbody>
</table>

**Month Names**

The following tokens are recognized as names of months:

<table>
<thead>
<tr>
<th>Month</th>
<th>Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>JANUARY</td>
<td>JAN</td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>FEB</td>
</tr>
<tr>
<td>Month</td>
<td>Abbreviations</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>MARCH</td>
<td>MAR</td>
</tr>
<tr>
<td>APRIL</td>
<td>APR</td>
</tr>
<tr>
<td>MAY</td>
<td>MAY</td>
</tr>
<tr>
<td>JUNE</td>
<td>JUN</td>
</tr>
<tr>
<td>JULY</td>
<td>JUL</td>
</tr>
<tr>
<td>AUGUST</td>
<td>AUG</td>
</tr>
<tr>
<td>SEPTEMBER</td>
<td>SEP, SEPT</td>
</tr>
<tr>
<td>OCTOBER</td>
<td>OCT</td>
</tr>
<tr>
<td>NOVEMBER</td>
<td>NOV</td>
</tr>
<tr>
<td>DECEMBER</td>
<td>DEC</td>
</tr>
</tbody>
</table>
Interval Literal

A literal that represents a time span.

Syntax

\[ @ \ [ - \{ \textit{quantity} \textit{subtype-unit} \} \ldots \] \ [ AGO ] \]

Parameters

<table>
<thead>
<tr>
<th>@</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (minus)</td>
<td>Specifies a negative interval value.</td>
</tr>
</tbody>
</table>

\textit{quantity}  Integer numeric constant

\textit{subtype-unit}  See Interval Subtype Units for valid values. Subtype units must be specified for year-month intervals; they are optional for day-time intervals.

AGO  Specifies a negative interval value. AGO and - (minus) are synonymous.

Notes

- The amounts of different units are implicitly added up with appropriate sign accounting.
- The boundaries of an interval constant are:
  - \( 9223372036854775807 \) usec to \(-9223372036854775807 \) usec
  - \( 296533 \) years \( 3 \) mons \( 21 \) days \( 04:00:54.775807 \) to \(-296533 \) years \(-3 \) mons \(-21 \) days \(-04:00:54.775807 \)
- The range of an interval constant is +/- \( 2^{63} - 1 \) (plus or minus two to the sixty-third minus one) microseconds.
- In Vertica, interval fields are additive and accept large floating-point numbers.

Examples

See Specifying Interval Input.Interval Literal
Interval Subtype Units

The following tables lists subtype units that can be specified in an interval literal, divided into major categories:

- **Year-month subtype units**
- **Day-time subtype units**

**Year-month subtype units**

<table>
<thead>
<tr>
<th>Subtypes</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Millennium</td>
<td>mil</td>
<td>millennium</td>
</tr>
<tr>
<td></td>
<td>millennia</td>
<td>mils</td>
</tr>
<tr>
<td>Century</td>
<td>c</td>
<td>cent</td>
</tr>
<tr>
<td>Decade</td>
<td>dec</td>
<td>decade</td>
</tr>
<tr>
<td>Year</td>
<td>a</td>
<td>year</td>
</tr>
<tr>
<td></td>
<td>ka</td>
<td>julian kilo-year: 365250 days</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>Calendar year: 365 days</td>
</tr>
<tr>
<td></td>
<td>yrs</td>
<td></td>
</tr>
<tr>
<td>Quarter</td>
<td>q</td>
<td>qtr</td>
</tr>
<tr>
<td>Month</td>
<td>mon</td>
<td>month</td>
</tr>
<tr>
<td></td>
<td>Vertica can interpret m as minute or month, depending on context. See Processing m Input below.</td>
<td></td>
</tr>
<tr>
<td>Week</td>
<td>w</td>
<td>week</td>
</tr>
</tbody>
</table>
## Day-time subtype units

<table>
<thead>
<tr>
<th>Subtypes</th>
<th>Units</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>d</td>
<td>day</td>
</tr>
<tr>
<td>Hour</td>
<td>h</td>
<td>hour</td>
</tr>
<tr>
<td>Minute</td>
<td>min</td>
<td>minute</td>
</tr>
<tr>
<td>Second</td>
<td>s</td>
<td>sec</td>
</tr>
<tr>
<td>Millisecond</td>
<td>millisecond</td>
<td>ms</td>
</tr>
<tr>
<td>Microsecond</td>
<td>microsecond</td>
<td>us</td>
</tr>
</tbody>
</table>
Processing m Input

Vertica uses context to interpret the input unit m as months or minutes. For example, the following command creates a one-column table with an interval value:

```sql
=> CREATE TABLE int_test(i INTERVAL YEAR TO MONTH);
```

Given the following INSERT statement, Vertica interprets the interval literal 1y 6m as 1 year 6 months:

```sql
=> INSERT INTO int_test VALUES('1y 6m');
     -------
     1
     (1 row)
```

```sql
=> SET INTERVALSTYLE TO UNITS;
=> SELECT * FROM int_test;
     -------
     i
     1 year 6 months
     (1 row)
```

The following `ALTER TABLE` statement adds a DAY TO MINUTE interval column to table `int_test`:

```sql
=> ALTER TABLE int_test ADD COLUMN x INTERVAL DAY TO MINUTE;
ALTER TABLE
```

The next INSERT statement sets the first and second columns to 3y 20m and 1y 6m, respectively. In this case, Vertica interprets the m input literals in two ways:

- For column i, Vertica interprets the m input as months, and displays 4 years 8 months.

- For column x, Vertica interprets the m input as minutes. Because the interval is defined as DAY TO MINUTE, it converts the inserted input value 1y 6m to 365 days 6 minutes:

```sql
=> INSERT INTO int_test VALUES('3y 20m', '1y 6m');
     ------
     1
     (1 row)
```

```sql
=> SELECT * FROM int_test;
     -------
     i  |   x
     1 year 6 months  | 365 days 6 mins
     (2 rows)
```
Interval Qualifier

Specifies how to interpret and format an interval literal for output, and, optionally, sets precision. Interval qualifiers are composed of one or two units:

\[ \text{unit}[p] \ [ \text{TO} \ \text{unit}[p] \ ] \]

where:

- \( \text{unit} \) specifies a day-time or year-month subtype.
- \( p \) specifies precision, an integer between 0 and 6. In general, precision only applies to \( \text{SECOND} \) units. The default precision for \( \text{SECOND} \) is 6. For details, see Specifying Interval Precision.

If an interval omits an interval qualifier, the default is \( \text{DAY TO SECOND(6)} \).

Interval qualifiers are divided into two categories: day-time and year-month, as shown in the tables below.

**Day-time interval qualifiers**

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>DAY TO HOUR</td>
<td>Span of days and hours</td>
</tr>
<tr>
<td>DAY TO MINUTE</td>
<td>Span of days and minutes</td>
</tr>
<tr>
<td>DAY TO SECOND</td>
<td>Span of days, hours, minutes, seconds, and fractions of a second.</td>
</tr>
<tr>
<td>HOUR</td>
<td>Hours within days</td>
</tr>
<tr>
<td>HOUR TO MINUTE</td>
<td>Span of hours and minutes</td>
</tr>
<tr>
<td>HOUR TO SECOND</td>
<td>Span of hours and seconds</td>
</tr>
<tr>
<td>MINUTE</td>
<td>Minutes within hours</td>
</tr>
<tr>
<td>MINUTE TO SECOND</td>
<td>Span of minutes and seconds</td>
</tr>
<tr>
<td>SECOND</td>
<td>Seconds within minutes</td>
</tr>
</tbody>
</table>

Note: The \( \text{SECOND} \) field can have an interval fractional seconds precision, which indicates the number of decimal digits maintained following the decimal point in the SECONDS.
## Day-time interval qualifiers, continued

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECOND</td>
<td>value. When <strong>SECOND</strong> is not the first field, it has a precision of 2 places before the decimal point.</td>
</tr>
</tbody>
</table>

## Year-month interval qualifiers

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>MONTH</td>
<td>Months within year</td>
</tr>
<tr>
<td>YEAR TO MONTH</td>
<td>Span of years and months</td>
</tr>
</tbody>
</table>

**Note:** Vertica also supports `INTERVALYM`, which is an alias for `INTERVAL YEAR TO MONTH`. Thus, the following two statements are equivalent:

```sql
=> SELECT INTERVALYM '1 2';
?column?----------------------
1 year 2 months

=> SELECT INTERVAL '1 2' YEAR TO MONTH;
?column?----------------------
1 year 2 months
(1 row)
```
Examples

See Controlling Interval Format.
Operators

Operators are logical, mathematical, and equality symbols used in SQL to evaluate, compare, or calculate values.

Bitwise Operators

Bitwise operators perform bit manipulations on INTEGER and BINARY/VARBINARY data types:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>AND</td>
<td>12 &amp; 4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR</td>
<td>32</td>
</tr>
<tr>
<td>#</td>
<td>XOR</td>
<td>17 # 5</td>
<td>20</td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
<td>~1</td>
<td>~2</td>
</tr>
<tr>
<td>&lt;&lt;†</td>
<td>Bitwise shift left</td>
<td>1 &lt;&lt; 4</td>
<td>16</td>
</tr>
<tr>
<td>&gt;&gt;†</td>
<td>Bitwise shift right</td>
<td>8 &gt;&gt; 2</td>
<td>2</td>
</tr>
</tbody>
</table>

† Invalid for BINARY/VARBINARY data types

String Argument Handling

String arguments must be explicitly cast as BINARY or VARBINARY data types for all bitwise operators. For example:

```sql
=> SELECT 'xyz':VARBINARY & 'zyx':VARBINARY AS AND;
     AND
    -----
   xyx
   (1 row)

=> SELECT 'xyz':VARBINARY | 'zyx':VARBINARY AS OR;
   OR
  -----
  zyz
  (1 row)
```
Bitwise operators treats all string arguments as equal in length. If the arguments have different lengths, the operator function right-pads the smaller string with one or more zero bytes to equal the length of the larger string.

For example, the following statement ANDs unequal strings *xyz* and *zy*. Vertica right-pads string *zy* with one zero byte. The last character in the result is represented accordingly, as \000:

```sql
=> SELECT 'xyz'::VARBINARY & 'zy'::VARBINARY AS AND;

--
xy\000
(1 row)
```

**Boolean Operators**

Vertica supports the following Boolean operators:

- **AND**
- **OR**
- **NOT**

Operators **AND** and **OR** are commutative, that is, you can switch left and right operands without affecting the result. However, the order of evaluation of sub-expressions is not defined. To force evaluation order, use a **CASE** construct.

**Caution:** Do not confuse Boolean operators with the [Boolean predicate](https://www.vertica.com/docs/9.x/Vertica-Documentations/sql-reference-manual/vertica-analytic-database-9.0.x-page-2132.html) or Boolean data type, which can have only two values: true and false.

**Logic**

SQL uses a three-valued Boolean logic where the NULL represents "unknown."

<table>
<thead>
<tr>
<th>If a =</th>
<th>and b =</th>
<th>then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a AND b =</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>
Comparison Operators

Comparison operators are available for all data types where comparison makes sense. All comparison operators are binary operators that return values of true, false, or NULL.

### Operator | Description | Binary function
--- | --- | ---
< | less than | binary_lt
> | greater than | binary_gt
<= | less than or equal to | binary_le
>= | greater than or equal to | binary_ge
= | equal | binary_eq
<> | not equal | binary_ne
NULL Handling

Comparison operators return NULL if either or both operands are null. One exception applies: <= returns true if both operands are NULL, and false if one operand is NULL.

Data Type Coercion Operators (CAST)

Data type coercion (casting) passes an expression value to an input conversion routine for a specified data type, resulting in a constant of the indicated type.

Syntax

SELECT CAST ( expression AS data_type )
SELECT expression::data_type
SELECT data_type 'string'

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>An expression of any type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_type</td>
<td>Converts the value of expression to one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>• BINARY</td>
</tr>
<tr>
<td></td>
<td>• BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>• CHARACTER</td>
</tr>
<tr>
<td></td>
<td>• DATE/TIME</td>
</tr>
<tr>
<td></td>
<td>• NUMERIC</td>
</tr>
</tbody>
</table>

Notes

- In Vertica, data type coercion (casting) can be invoked by an explicit cast request. It must use one of the following constructs:
The explicit type cast can be omitted if there is no ambiguity as to the type the constant
must be. For example, when a constant is assigned directly to a column, it is automatically
coerced to the column's data type.

If a binary value is cast (implicitly or explicitly) to a binary type with a smaller length, the
value is silently truncated. For example:

```sql
=> SELECT 'abcd'::BINARY(2);
?column?
-------
ab
(1 row)
```

Similarly, if a character value is cast (implicitly or explicitly) to a character value with a
smaller length, the value is silently truncated. For example:

```sql
=> SELECT 'abcd'::CHAR(3);
?column?
-------
abc
```

Vertica supports only casts and resize operations as follows:

- **BINARY** to and from **VARBINARY**
- **VARBINARY** to and from **LONG VARBINARY**
- **BINARY** to and from **LONG VARBINARY**

On binary data that contains a value with fewer bytes than the target column, values are
right-extended with the zero byte '\0' to the full width of the column. Trailing zeros on
variable-length binary values are not right-extended:

```sql
=> SELECT 'ab'::BINARY(4), 'ab'::VARBINARY(4), 'ab'::LONG VARBINARY(4);
--------------------------
ab\000\000 | ab | ab
(1 row)
```
Examples

```sql
=> SELECT CAST((2 + 2) AS VARCHAR);
?column?
---------
 4
(1 row)
```

```sql
=> SELECT (2 + 2)::VARCHAR;
?column?
---------
 4
(1 row)
```

```sql
=> SELECT INTEGER '123';
?column?
---------
 123
(1 row)
```

```sql
=> SELECT (2 + 2)::LONG VARCHAR
?column?
---------
 4
(1 row)
```

```sql
=> SELECT '2.2' + 2;
   ERROR: invalid input syntax for integer: "2.2"
```

```sql
=> SELECT FLOAT '2.2' + 2;
?column?
---------
 4.2
(1 row)
```

See Also

- Data Type Conversions
- Data Type Coercion Chart
- CAST Failures

Cast Failures

When you invoke data type coercion (casting) by an explicit cast and the cast fails, the result returns either an error or NULL. Cast failures commonly occur when you attempt to cast
conflicting conversions, such as trying to convert a varchar expression that contains letters to an integer.

When a cast fails, the result returned depends on the data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Cast Failure Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/Time</td>
<td>NULL</td>
</tr>
<tr>
<td>Literal</td>
<td>Error</td>
</tr>
<tr>
<td>All Other Types</td>
<td>Error</td>
</tr>
</tbody>
</table>

Enabling Strict Time Casts

You can enable all cast failures to result in an error, including those for Date/Time data types. Doing so allows you to see the reason why some or all of the cast failed. To return an error instead of NULL, use the `ALTER SESSION` statement with the SET parameter:

```
ALTER SESSION SET EnableStrictTimeCasts=1;
```

The following example shows a Date/Time cast failure that returns NULL:

```
=> CREATE TABLE mytable (a VARCHAR);
CREATE TABLE
=> INSERT INTO mytable VALUES('string');
OUTPUT
-------
1
(1 row)
=> INSERT INTO mytable VALUES('1');
OUTPUT
-------
1
(1 row)
=> SELECT a::time FROM mytable;
a
---
(2 rows)
```

When you specify EnableStrictTimeCasts, the cast failure returns an error:

```
=> ALTER SESSION SET EnableStrictTimeCasts=1;
ALTER SESSION
=> SELECT a::time FROM mytable;
ERROR 2005:  Invalid input syntax for time: "1"
```
Returning All Cast Failures as NULL

To explicitly cast an expression to a requested data type, use the following construct:

```
SELECT expression::data_type
```

Using this command to cast any values to a conflicting data type returns the following error:

```
ERROR 2827: Could not convert "string" from column table.a to an int8
```

In addition to the `::` cast, Vertica supports the use of `::!`. Use `::!` instead of `::`, if you want to return:

- NULL instead of an error for any non-Date/Time data types
- NULL instead of an error after setting EnableStrictTimeCasts

Returning all cast failures as NULL allows those expressions that succeeded during the cast to appear in the result. Those expressions which failed during the cast, however, have a NULL value.

The following example shows a cast failure that returns an error:

```
=> CREATE TABLE mytable (a VARCHAR);
CREATE TABLE
=> INSERT INTO mytable VALUES('string');
OUTPUT
-------
1
(1 row)
=> INSERT INTO mytable VALUES('1');
OUTPUT
-------
1
(1 row)
=> SELECT a::int FROM mytable;
ERROR 2827: Could not convert "string" from column mytable.a to an int8
```

When you use `::!`, the cast fails for the "string" value and returns NULL. However, it succeeds for the "1" value and returns 1:

```
=> SELECT a::!int FROM mytable;
a
---
1
(2 rows)
```
Date/Time Operators

Syntax

[ + | - | * | / ]

Parameters

+  Addition
-  Subtraction
*  Multiplication
/  Division

Notes

- The operators described below that take TIME or TIMESTAMP inputs actually come in two variants: one that takes TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, and one that takes TIME WITHOUT TIME ZONE or TIMESTAMP WITHOUT TIME ZONE. For brevity, these variants are not shown separately.

- The + and * operators come in commutative pairs (for example both DATE + INTEGER and INTEGER + DATE); only one of each such pair is shown.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result Type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE '2001-09-28' + INTEGER '7'</td>
<td>DATE</td>
<td>'2001-10-05'</td>
</tr>
<tr>
<td>DATE '2001-09-28' + INTERVAL '1 HOUR'</td>
<td>TIMESTAMP</td>
<td>'2001-09-28 01:00:00'</td>
</tr>
<tr>
<td>DATE '2001-09-28' + TIME '03:00'</td>
<td>TIMESTAMP</td>
<td>'2001-09-28 03:00:00'</td>
</tr>
<tr>
<td>INTERVAL '1 DAY' + INTERVAL '1 HOUR'</td>
<td>INTERVAL</td>
<td>'1 DAY 01:00:00'</td>
</tr>
<tr>
<td>TIMESTAMP '2001-09-28 01:00:00' + INTERVAL '23 HOURS'</td>
<td>TIMESTAMP</td>
<td>'2001-09-29 00:00:00'</td>
</tr>
<tr>
<td>TIME '01:00' + INTERVAL '3 HOURS'</td>
<td>TIME</td>
<td>'04:00:00'</td>
</tr>
<tr>
<td>- INTERVAL '23 HOURS'</td>
<td>INTERVAL</td>
<td>'-23:00:00'</td>
</tr>
<tr>
<td>DATE '2001-10-01' - DATE</td>
<td>INTEGER</td>
<td>'3'</td>
</tr>
</tbody>
</table>
### Mathematical Operators

Mathematical operators are provided for many data types.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Factorial</td>
<td>$5!$</td>
<td>120</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>$2 + 3$</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>$2 - 3$</td>
<td>-1</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>$2 * 3$</td>
<td>6</td>
</tr>
<tr>
<td>/</td>
<td>Division (integer division produces NUMERIC results)</td>
<td>$4 / 2$</td>
<td>2.00...</td>
</tr>
<tr>
<td>//</td>
<td>With integer division, returns an INTEGER rather</td>
<td>$117.32 // 2.5$</td>
<td>46</td>
</tr>
</tbody>
</table>
### Factorial Operator Support

Vertica supports use of factorial operators on positive and negative floating point (DOUBLE PRECISION) numbers and integers. For example:

```
=> SELECT 4.98!;
   ?column?
-------------------------
  115.978600750905
(1 row)
```

Factorial is defined in terms of the gamma function, where \((-1) = \infty\) and the other negative integers are undefined. For example:

\[
(-4)! = \text{NaN} \\
-(4!) = -24
\]

Factorial is defined as follows for all complex numbers \(z\):

\[z! = \text{gamma}(z+1)\]

For details, see Abramowitz and Stegun: Handbook of Mathematical Functions.

### NULL Operators

To check whether a value is or is not NULL, use the constructs:

```
[expression IS NULL | expression IS NOT NULL]
```

Alternatively, use equivalent, but nonstandard, constructs:
Do not write $expression = NULL$ because NULL represents an unknown value, and two unknown values are not necessarily equal. This behavior conforms to the SQL standard.

Note: Some applications might expect that $expression = NULL$ returns true if $expression$ evaluates to null. Vertica strongly recommends that these applications be modified to comply with the SQL standard.

String Concatenation Operators

To concatenate two strings on a single line, use the concatenation operator (two consecutive vertical bars).

**Syntax**

```
string || string
```

**Parameters**

| $string$ | Is an expression of type CHAR or VARCHAR |

**Notes**

- $||$ is used to concatenate expressions and constants. The expressions are cast to VARCHAR if possible, otherwise to VARBINARY, and must both be one or the other.
- Two consecutive strings within a single SQL statement on separate lines are automatically concatenated

**Examples**

The following example is a single string written on two lines:

```
=> SELECT E'xx' || '\\';
?column?
----------
```
The following examples show two strings concatenated:

```sql
=> SELECT E'xx' || '\n';
?column?
--------
xx\n
(1 row)

=> SELECT 'auto' || 'mobile';
?column?
---------
automobile

(1 row)

=> SELECT 'auto'-> 'mobile';
?column?
---------
automobile

(1 row)

=> SELECT 1 || 2;
?column?
--------
12

(1 row)

=> SELECT '1' || '2';
?column?
--------
12

(1 row)

=> SELECT '1'-> '2';
?column?
---------
12

(1 row)
```
Expressions

SQL expressions are the components of a query that compare a value or values against other values. They can also perform calculations. Expressions found inside any SQL command are usually in the form of a conditional statement.

Operator Precedence

The following table shows operator precedence in decreasing (high to low) order.

<table>
<thead>
<tr>
<th>Operator/Element</th>
<th>Associativity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>left</td>
<td>table/column name separator</td>
</tr>
<tr>
<td>::</td>
<td>left</td>
<td>typecast</td>
</tr>
<tr>
<td>[ ]</td>
<td>left</td>
<td>array element selection</td>
</tr>
<tr>
<td>-</td>
<td>right</td>
<td>unary minus</td>
</tr>
<tr>
<td>^</td>
<td>left</td>
<td>exponentiation</td>
</tr>
<tr>
<td>* / %</td>
<td>left</td>
<td>multiplication, division, modulo</td>
</tr>
<tr>
<td>+ -</td>
<td>left</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>IS</td>
<td></td>
<td>IS TRUE, IS FALSE, IS UNKNOWN, IS NULL</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>set membership</td>
</tr>
<tr>
<td>BETWEEN</td>
<td></td>
<td>range containment</td>
</tr>
<tr>
<td>OVERLAPS</td>
<td></td>
<td>time interval overlap</td>
</tr>
<tr>
<td>LIKE</td>
<td></td>
<td>string pattern matching</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td></td>
<td>less than, greater than</td>
</tr>
</tbody>
</table>

Note: When an expression includes more than one operator, Vertica recommends that you specify the order of operation using parentheses, rather than relying on operator precedence.
<table>
<thead>
<tr>
<th>Operator/Element</th>
<th>Associativity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>right</td>
<td>equality, assignment</td>
</tr>
<tr>
<td>NOT</td>
<td>right</td>
<td>logical negation</td>
</tr>
<tr>
<td>AND</td>
<td>left</td>
<td>logical conjunction</td>
</tr>
<tr>
<td>OR</td>
<td>left</td>
<td>logical disjunction</td>
</tr>
</tbody>
</table>

### Expression Evaluation Rules

The order of evaluation of subexpressions is not defined. In particular, the inputs of an operator or function are not necessarily evaluated left-to-right or in any other fixed order. To force evaluation in a specific order, use a CASE construct. For example, this is an untrustworthy way of trying to avoid division by zero in a WHERE clause:

```
=> SELECT x, y WHERE x <> 0 AND y/x > 1.5;
```

But this is safe:

```
=> SELECT x, y
WHERE
CASE
  WHEN x <> 0 THEN y/x > 1.5
  ELSE false
END;
```

A CASE construct used in this fashion defeats optimization attempts, so use it only when necessary. (In this particular example, it would be best to avoid the issue by writing `y > 1.5*x` instead.)

### Limits to SQL Expressions

There are some limits on the number of modifiers and recursions that you can make in an expression. There are two limits that you should be aware of:

- The first limit is based on the stack available to the expression. [[[Undefined variable Vertica.DBMS_UPPERCASE]]] requires at least 100kb of free stack. If this limit is exceeded then the error "The query contains an expression that is too complex to analyze" may be thrown. Adding additional physical memory and/or increasing the value of `ulimit -s max` increase the available stack and prevent the error.
The second limit is the number of recursions possible in an analytic expression. The limit is 2000. If this limit is exceeded then the error "The query contains an expression that is too complex to analyze" may be thrown. This limit cannot be increased.

Aggregate Expressions

An aggregate expression applies an aggregate function across the rows or groups of rows selected by a query.

An aggregate expression only can appear in the select list or HAVING clause of a SELECT statement. It is invalid in other clauses such as WHERE, because those clauses are evaluated before the results of aggregates are formed.

Syntax

An aggregate expression has the following format:

```
aggregate-function ( [ * ] [ ALL | DISTINCT ] expression )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aggregate-function</code></td>
<td>A Vertica function that aggregates data over groups of rows from a query result set.</td>
</tr>
<tr>
<td>`ALL</td>
<td>DISTINCT`</td>
</tr>
<tr>
<td></td>
<td>• ALL (default): Invokes <code>aggregate-function</code> across all input rows where <code>expression</code> evaluates to a non-null value.</td>
</tr>
<tr>
<td></td>
<td>• DISTINCT: Invokes <code>aggregate-function</code> across all input rows where <code>expression</code> evaluates to a unique non-null value.</td>
</tr>
<tr>
<td><code>expression</code></td>
<td>A value expression that does not itself contain an aggregate expression.</td>
</tr>
</tbody>
</table>
CASE Expressions

The CASE expression is a generic conditional expression that can be used wherever an expression is valid. It is similar to case and if/then/else statements in other languages.

Syntax (form 1)

```
CASE
  WHEN condition THEN result
  [ WHEN condition THEN result ]
  ...
  [ ELSE result ]
END
```

Parameters

<table>
<thead>
<tr>
<th>condition</th>
<th>Is an expression that returns a boolean (true/false) result. If the result is false, subsequent WHEN clauses are evaluated in the same manner.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Specifies the value to return when the associated condition is true.</td>
</tr>
<tr>
<td>ELSE result</td>
<td>If no condition is true then the value of the CASE expression is the result in the ELSE clause. If the ELSE clause is omitted and no condition matches, the result is null.</td>
</tr>
</tbody>
</table>

Syntax (form 2)

```
CASE expression
  WHEN value THEN result
  [ WHEN value THEN result ]
  ...
  [ ELSE result ]
END
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>An expression that is evaluated and compared to all the value specifications in the WHEN clauses until one is found that is equal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies a value to compare to the expression.</td>
</tr>
<tr>
<td>result</td>
<td>Specifies the value to return when the expression is equal to the specified value.</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ELSE result</td>
<td>Specifies the value to return when the expression is not equal to any value; if no ELSE clause is specified, the value returned is null.</td>
</tr>
</tbody>
</table>

**Notes**

The data types of all the result expressions must be convertible to a single output type.

**Examples**

The following examples show two uses of the CASE statement.

```sql
=> SELECT * FROM test;
a
---
1
2
3
=> SELECT a,
    CASE WHEN a=1 THEN 'one'
         WHEN a=2 THEN 'two'
         ELSE 'other'
    END
FROM test;
a | case
----
1 | one
2 | two
3 | other
=> SELECT a,
    CASE a WHEN 1 THEN 'one'
         WHEN 2 THEN 'two'
         ELSE 'other'
    END
FROM test;
a | case
----
1 | one
2 | two
3 | other
```

**Special Example**

A CASE expression does not evaluate subexpressions that are not needed to determine the result. You can use this behavior to avoid division-by-zero errors:
Column References

Syntax

```sql
[[[database.]schema.]table-name.]column-name
```

Parameters

<table>
<thead>
<tr>
<th>schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>table-name</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• The name of a table</td>
</tr>
<tr>
<td></td>
<td>• A table alias defined in the query's FROM clause</td>
</tr>
<tr>
<td>column-name</td>
<td>A column name that is unique among all queried tables.</td>
</tr>
</tbody>
</table>

Restrictions

A a column reference cannot contain any spaces.

Comments

A comment is an arbitrary sequence of characters beginning with two consecutive hyphen characters and extending to the end of the line. For example:

```sql
-- This is a standard SQL comment
```
A comment is removed from the input stream before further syntax analysis and is effectively replaced by white space.

Alternatively, C-style block comments can be used where the comment begins with /* and extends to the matching occurrence of */.

```
/* multiline comment
 * with nesting: /* nested block comment */
 */
```

These block comments nest, as specified in the SQL standard. Unlike C, you can comment out larger blocks of code that might contain existing block comments.

---

**Date/Time Expressions**

Vertica uses an internal heuristic parser for all date/time input support. Dates and times are input as strings, and are broken up into distinct fields with a preliminary determination of what kind of information might be in the field. Each field is interpreted and either assigned a numeric value, ignored, or rejected. The parser contains internal lookup tables for all textual fields, including months, days of the week, and time zones.

Vertica parses date/time type inputs as follows:

1. Break the input string into tokens and categorize each token as a string, time, time zone, or number.

2. Numeric token contains:
   - colon (:)— Parse as a time string, include all subsequent digits and colons.
   - dash (-), slash (/), or two or more dots (.)— Parse as a date string which might have a text month.
   - Numeric only — Parse as a single field or an ISO 8601 concatenated date (19990113 for January 13, 1999) or time (141516 for 14:15:16).

3. Token starts with a plus (+) or minus (–): Parse as a time zone or a special field.

4. Token is a text string: match up with possible strings.
   - Perform a binary-search table lookup for the token as either a special string (for example, today), day (for example, Thursday), month (for example, January), or noise word (for example, at, on).
   - Set field values and bit mask for fields. For example, set year, month, day for today, and additionally hour, minute, second for now.
If not found, do a similar binary-search table lookup to match the token with a time zone.

If still not found, throw an error.

5. Token is a number or number field:
   - If eight or six digits, and if no other date fields were previously read, interpret as a "concatenated date" (19990118 or 990118). The interpretation is YYYYMMDD or YYMMDD.
   - If token is three digits and a year was already read, interpret as day of year.
   - If four or six digits and a year was already read, interpret as a time (HHMM or HHMMSS).
   - If three or more digits and no date fields were found yet, interpret as a year (this forces yy-mm-dd ordering of the remaining date fields).
   - Otherwise the date field ordering is assumed to follow the DateStyle setting: mm-dd-yy, dd-mm-yy, or yy-mm-dd. Throw an error if a month or day field is found to be out of range.

6. If BC is specified: negate the year and add one for internal storage. (In the Vertica implementation, 1 BC = year zero.)

7. If BC is not specified, and year field is two digits in length: adjust the year to four digits. If field is less than 70, add 2000, otherwise add 1900.

   Tip: Gregorian years AD 1–99 can be entered as 4 digits with leading zeros— for example, 0099 = AD 99.

Month Day Year Ordering

For some formats, ordering of month, day, and year in date input is ambiguous and there is support for specifying the expected ordering of these fields.

Special Date/Time Values

Vertica supports several special date/time values for convenience, as shown below. All of these values need to be written in single quotes when used as constants in SQL statements.

The values INFINITY and -INFINITY are specially represented inside the system and are displayed the same way. The others are simply notational shorthands that are converted to
ordinary date/time values when read. (In particular, NOW and related strings are converted to a specific time value as soon as they are read.)

<table>
<thead>
<tr>
<th>String</th>
<th>Valid Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epoch</td>
<td>DATE, TIMESTAMP</td>
<td>1970-01-01 00:00:00+00 (UNIX SYSTEM TIME ZERO)</td>
</tr>
<tr>
<td>INFINITY</td>
<td>TIMESTAMP</td>
<td>Later than all other time stamps</td>
</tr>
<tr>
<td>-INFINITY</td>
<td>TIMESTAMP</td>
<td>Earlier than all other time stamps</td>
</tr>
<tr>
<td>NOW</td>
<td>DATE, TIME, TIMESTAMP</td>
<td>Current transaction's start time</td>
</tr>
<tr>
<td>TODAY</td>
<td>DATE, TIMESTAMP</td>
<td>Midnight today</td>
</tr>
<tr>
<td>TOMORROW</td>
<td>DATE, TIMESTAMP</td>
<td>Midnight tomorrow</td>
</tr>
<tr>
<td>YESTERDAY</td>
<td>DATE, TIMESTAMP</td>
<td>Midnight yesterday</td>
</tr>
<tr>
<td>ALLBALLS</td>
<td>TIME</td>
<td>00:00:00.00 UTC</td>
</tr>
</tbody>
</table>

The following SQL-compatible functions can also be used to obtain the current time value for the corresponding data type:

- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- LOCALTIME
- LOCLTIMESTAMP

The latter four accept an optional precision specification. (See Date/Time Functions.) However, these functions are SQL functions and are not recognized as data input strings.

**NULL Value**

NULL is a reserved keyword used to indicate that a data value is unknown.

Be very careful when using NULL in expressions. NULL is not greater than, less than, equal to, or not equal to any other expression. Use the Boolean-Predicate for determining whether an expression value is NULL.
Notes

- Vertica stores data in projections, which are sorted in a specific way. All columns are stored in ASC (ascending) order. For columns of data type NUMERIC, INTEGER, DATE, TIME, TIMESTAMP, and INTERVAL, NULL values are placed at the beginning of sorted projections (NULLS FIRST), while for columns of data type FLOAT, STRING, and BOOLEAN, NULL values are placed at the end (NULLS LAST). For details, see NULL Sort Order and Runtime Sorting of NULL Values in Analytic Functions in Analyzing Data.

- Vertica also accepts NULL characters (\0) in constant strings and no longer removes null characters from VARCHAR fields on input or output. NULL is the ASCII abbreviation for the NULL character.

- You can write queries with expressions that contain the <=> operator for NULL=NULL joins. See Equi-Joins and Non Equi-Joins in Analyzing Data.

See Also

NULL-handling Functions
Predicates

Predicates are truth-tests. If the predicate test is true, it returns a value. Each predicate is evaluated per row, so that when the predicate is part of an entire table SELECT statement, the statement can return multiple results.

Predicates consist of a set of parameters and arguments. For example, in the following example WHERE clause:

```
WHERE name = 'Smith';
```

- `name = 'Smith'` is the predicate
- `'Smith'` is an expression

BETWEEN-predicate

The special BETWEEN predicate is available as a convenience.

Syntax

```
WHERE a BETWEEN x AND y
```

Examples

```
WHERE a BETWEEN x AND y
```

is equivalent to:

```
WHERE a >= x AND a <= y
```

Similarly:

```
WHERE a NOT BETWEEN x AND y
```

is equivalent to:

```
WHERE a < x OR a > y
```

You can use the BETWEEN predicate for date ranges:
CREATE TABLE t1 (c1 INT, c2 INT, c3 DATE);
COPY t1 FROM stdin DELIMITER ' | ';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

> 1 | 2 | 2014-07-26
> 2 | 3 | 2014-07-27
> 3 | 4 | 2014-07-28
> 4 | 5 | 2014-07-29
> 5 | 6 | 2014-07-30
> 6 | 7 | 2014-07-31
> 7 | 8 | 2014-08-01
> 8 | 9 | 2014-08-02
> 
> SELECT* FROM t1 WHERE c3 BETWEEN DATE('2014-07-26') AND DATE('2014-07-30');

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2014-07-26</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2014-07-27</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2014-07-28</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2014-07-29</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2014-07-30</td>
</tr>
</tbody>
</table>

(5 rows)

You can also use the NOW and INTERVAL keywords to select from a date range:

SELECT* FROM t1 WHERE c3 BETWEEN NOW()-INTERVAL '1 week' AND NOW();

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>2014-08-01</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2014-07-26</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2014-07-27</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2014-07-28</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2014-07-29</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2014-07-30</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2014-07-31</td>
</tr>
</tbody>
</table>

(7 rows)

### Boolean-Predicate

Retrieves rows where the value of an expression is true, false, or unknown (null).

### Syntax

- `expression` IS [NOT] TRUE
- `expression` IS [NOT] FALSE
- `expression` IS [NOT] UNKNOWN
Notes

- A null input is treated as the value UNKNOWN.

- IS UNKNOWN and IS NOT UNKNOWN are effectively the same as the **NULL-predicate**, except that the input expression does not have to be a single column value. To check a single column value for NULL, use the NULL-predicate.

- Do not confuse the boolean-predicate with [Boolean Operators](#) or the [Boolean data type](#), which can have only two values: true and false.

Column-Value-Predicate

Syntax

```
column-name comparison-op constant-expression
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column-name</td>
<td>A single column of one the tables specified in the FROM Clause.</td>
</tr>
<tr>
<td>comparison-op</td>
<td>A <a href="#">Comparison Operators</a>.</td>
</tr>
<tr>
<td>constant-expression</td>
<td>A constant value of the same data type as the column-name.</td>
</tr>
</tbody>
</table>

Notes

To check a column value for NULL, use the **NULL-predicate**.

Examples

```
table.column1 = 2
table.column2 = 'Seafood'
table.column3 IS NULL
```
IN-predicate

Syntax

\[(column-list) \text{ [ NOT ]} \text{ IN ( values-list )}\]

Parameters

<table>
<thead>
<tr>
<th>\textit{column-list}</th>
<th>One or more comma-delimited columns in the queried tables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{values-list}</td>
<td>Comma-delimited list of constant values to find in the column-list columns. Each \textit{values-list} value maps to a column-list column according to their order in \textit{values-list} and \textit{column-list}, respectively. Column/value pairs must have \textit{compatible} data types. You can specify multiple sets of values as follows: (\left(\text{\textit{values-list}}, \text{\textit{values-list}}\right), \ldots)</td>
</tr>
</tbody>
</table>

Examples

The following \texttt{SELECT} statement queries all data in table \texttt{t11}.

\[
\texttt{=> SELECT * FROM t11 ORDER BY pk;}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{pk} & \text{col1} & \text{col2} & \text{SKIP\_ME\_FLAG} \\
\hline
1 & 2 & 3 & t \\
2 & 3 & 4 & t \\
3 & 4 & 5 & f \\
4 & 5 & 6 & f \\
5 & 6 & 7 & t \\
6 & 8 & 8 & f \\
7 & 8 & 9 & t \\
\hline
\end{array}
\]

(7 rows)

The following query specifies an IN predicate, to find all rows in \texttt{t11} where columns \texttt{col1} and \texttt{col2} contain values of \((2, 3)\) or \((6, 7)\):

\[
\texttt{=> SELECT * FROM t11 WHERE (col1, col2) IN ((2,3), (6,7)) ORDER BY pk;}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{pk} & \text{col1} & \text{col2} & \text{SKIP\_ME\_FLAG} \\
\hline
\end{array}
\]

(7 rows)
INTERPOLATE

Used to join two event series together using some ordered attribute, event series joins let you compare values from two series directly, rather than having to normalize the series to the same measurement interval.

Syntax

\[ \text{expression1 INTERPOLATE PREVIOUS VALUE expression2} \]

Parameters

| expression1 | A column reference from one the tables specified in the FROM Clause. The referenced columns are typically a DATE/TIME data type, often TIMESTAMP, inasmuch as you are joining data that represents an event series; however, the referenced columns can be of any type. |
| expression2 |
| PREVIOUS VALUE | Pads the non-preserved side with the previous values from relation when there is no match. Input rows are sorted in ascending logical order of the join column. |

Note: An ORDER BY clause, if used, does not determine the input order but only determines query output order.

Description

- An event series join is an extension of a regular outer join. Instead of padding the non-preserved side with null values when there is no match, the event series join pads the non-preserved side with the previous values from the table.
The difference between expressing a regular outer join and an event series join is the INTERPOLATE predicate, which is used in the ON clause. See the Examples section below Notes and Restrictions. See also Event Series Joins in Analyzing Data.

Data is logically partitioned on the table in which it resides, based on other ON clause equality predicates.

Interpolated values come from the table that contains the null, not from the other table.

Vertica does not guarantee that there will be no null values in the output. If there is no previous value for a mismatched row, that row will be padded with nulls.

Event series join requires that both tables be sorted on columns in equality predicates, in any order, followed by the INTERPOLATED column. If data is already sorted in this order, then an explicit sort is avoided, which can improve query performance. For example, given the following tables:

```
ask: exchange, stock, ts, price
bid: exchange, stock, ts, price
```

In the query that follows

- `ask` is sorted on `exchange`, `stock` (or the reverse), `ts`
- `bid` is sorted on `exchange`, `stock` (or the reverse), `ts`

```
SELECT ask.price - bid.price, ask.ts, ask.stock, ask.exchange
FROM ask FULL OUTER JOIN bid
ON ask.stock = bid.stock AND ask.exchange = bid.exchange AND ask.ts INTERPOLATE PREVIOUS VALUE bid.ts;
```

**Restrictions**

- Only one INTERPOLATE expression is allowed per join.
- INTERPOLATE expressions are used only with ANSI SQL-99 syntax (the ON clause), which is already true for full outer joins.
- INTERPOLATE can be used with equality predicates only.
- The AND operator is supported but not the OR and NOT operators.
- Expressions and implicit or explicit casts are not supported, but subqueries are allowed.
Example

The examples that follow use this simple schema.

```sql
CREATE TABLE t(x TIME);
CREATE TABLE t1(y TIME);
INSERT INTO t VALUES('12:40:23');
INSERT INTO t VALUES('14:40:25');
INSERT INTO t VALUES('14:45:00');
INSERT INTO t VALUES('14:49:55');
INSERT INTO t1 VALUES('12:40:23');
INSERT INTO t1 VALUES('14:00:00');
COMMIT;
```

Normal Full Outer Join

```sql
=> SELECT * FROM t FULL OUTER JOIN t1 ON t.x = t1.y;
```

Notice the null rows from the non-preserved table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
<tr>
<td>14:40:25</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:45:00</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:49:55</td>
<td>14:00:00</td>
</tr>
<tr>
<td></td>
<td>14:00:00</td>
</tr>
</tbody>
</table>

(5 rows)

Full Outer Join with Interpolation

```sql
=> SELECT * FROM t FULL OUTER JOIN t1 ON t.x INTERPOLATE
   PREVIOUS VALUE t1.y;
```

In this case, the rows with no entry point are padded with values from the previous row.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
<tr>
<td>12:40:23</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:40:25</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:45:00</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:49:55</td>
<td>14:00:00</td>
</tr>
<tr>
<td></td>
<td>14:00:00</td>
</tr>
</tbody>
</table>

(5 rows)
Normal Left Outer Join

=> SELECT * FROM t LEFT OUTER JOIN t1 ON t.x = t1.y;

Again, there are nulls in the non-preserved table

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
<tr>
<td>14:40:25</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:45:00</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:49:55</td>
<td>14:00:00</td>
</tr>
</tbody>
</table>

(4 rows)

Left Outer Join with Interpolation

=> SELECT * FROM t LEFT OUTER JOIN t1 ON t.x INTERPOLATE PREVIOUS VALUE t1.y;

Nulls padded with interpolated values.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
<tr>
<td>14:40:25</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:45:00</td>
<td>14:00:00</td>
</tr>
<tr>
<td>14:49:55</td>
<td>14:00:00</td>
</tr>
</tbody>
</table>

(4 rows)

Inner Joins

For inner joins, there is no difference between a regular inner join and an event series inner join. Since null values are eliminated from the result set, there is nothing to interpolate.

A regular inner join returns only the single matching row at 12:40:23:

=> SELECT * FROM t INNER JOIN t1 ON t.x = t1.y;

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
</tbody>
</table>

(1 row)

An event series inner join finds the same single-matching row at 12:40:23:
SELECT *
FROM t
INNER JOIN t1
ON t.x INTERPOLATE PREVIOUS VALUE t1.y;

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:40:23</td>
<td>12:40:23</td>
</tr>
</tbody>
</table>

(1 row)

Semantics

When you write an event series join in place of normal join, values are evaluated as follows (using the schema in the above examples):

- t is the outer, preserved table
- t1 is the inner, non-preserved table
- For each row in outer table t, the ON clause predicates are evaluated for each combination of each row in the inner table t1.
- If the ON clause predicates evaluate to true for any combination of rows, those combination rows are produced at the output.
- If the ON clause is false for all combinations, a single output row is produced with the values of the row from t along with the columns of t1 chosen from the row in t1 with the greatest t1.y value such that t1.y < t.x; if no such row is found, pad with nulls.

Note: t LEFT OUTER JOIN t1 is equivalent to t1 RIGHT OUTER JOIN t.

In the case of a full outer join, all values from both tables are preserved.

See Also

- Event Series Joins

Join-Predicate

Specifies the columns on which records from two or more tables are joined. You can connect multiple join predicates with logical operators AND, OR, and NOT.
Syntax

ON column-ref = column-ref [ {AND | OR | NOT} column-ref = column-ref ]...

Parameters

| column-ref | Specifies a column in a queried table. For best performance, do not join on LONG VARBINARY and LONG VARCHAR columns. |

LIKE-predicate

Retrieves rows where the string value of a column matches a specified pattern. The pattern can contain one or more wildcard characters.

Syntax

string-expression [ NOT ] { LIKE | ILIKE | LIKEB | ILIKEB } ‘pattern’ [ ESCAPE ‘escape-character’ ]

Parameters

<table>
<thead>
<tr>
<th>string-expression</th>
<th>The column values to search for pattern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>Returns true if LIKE returns false, and the reverse; equivalent to NOT string LIKE pattern.</td>
</tr>
<tr>
<td>pattern</td>
<td>Specifies what strings to match, typically containing one or both of the following wildcard characters:</td>
</tr>
<tr>
<td></td>
<td>- Underscore (_) matches any single character.</td>
</tr>
<tr>
<td></td>
<td>- Percent sign (%) matches any string of zero or more characters.</td>
</tr>
<tr>
<td>ESCAPE escape-character</td>
<td>Specifies an escape character, used in the to escape reserved characters underscore (_), percent (%), and the escape character itself. This is enforced only for non-default collations.</td>
</tr>
<tr>
<td></td>
<td>If you omit this parameter, you can use Vertica's default escape character.</td>
</tr>
</tbody>
</table>
character, backslash (\), which is valid for CHAR and VARCHAR strings.

Note: Backslash is not valid for binary data types character. To embed an escape character for binary data types, use ESCAPE to specify a valid binary character.

Substitute Symbols

You can substitute the following symbols for LIKE and its variants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>~#</td>
<td>LIKEB</td>
</tr>
<tr>
<td>~~~*</td>
<td>ILIKE</td>
</tr>
<tr>
<td>~##*</td>
<td>ILIKEB</td>
</tr>
<tr>
<td>!~~</td>
<td>NOT LIKE</td>
</tr>
<tr>
<td>!~#</td>
<td>NOT LIKEB</td>
</tr>
<tr>
<td>!~~*</td>
<td>NOT ILIKE</td>
</tr>
<tr>
<td>!~~##</td>
<td>NOT ILIKEB</td>
</tr>
</tbody>
</table>

Note: ESCAPE is not valid for the above symbols.

Pattern Matching

LIKE requires that the entire string expression match the pattern. To match a sequence of characters anywhere within a string, the pattern must start and end with a percent sign.

LIKE does not ignore trailing white space characters. If the data values to match end with an indeterminate amount of white space, append the wildcard character % to pattern.

LIKE Variants Compared

The LIKE predicate is compliant with the SQL standard. Vertica also supports several non-standard variants, notably ILIKE, which is equivalent to LIKE except it performs case-insensitive searches. The following differences pertain to LIKE and its variants:
• LIKE operates on UTF-8 character strings. Exact behavior depends on collation parameters such as strength. In particular, ILIKE works by setting S=2 (ignore case) in the current session locale.

• LIKE and ILIKE are stable for character strings, but immutable for binary strings, while LIKEB and ILIKEB are immutable for both cases.

• LIKEB and ILIKEB predicates do byte-at-a-time ASCII comparisons.

Locale Dependencies

In the default locale, LIKE and ILIKE handle UTF-8 character-at-a-time, locale-insensitive comparisons. ILIKE handles language-independent case-folding.

In non-default locales, LIKE and ILIKE perform locale-sensitive string comparisons, including some automatic normalization, using the same algorithm as the "=" operator on VARCHAR types.

ESCAPE expressions evaluate to exactly one octet—or one UTF-8 character for non-default locales.

Examples

The following example illustrates pattern matching in locales.

```sql
locale default=> CREATE TABLE src(c1 VARCHAR(100));
=> INSERT INTO src VALUES (U&'\00DF'); --The sharp ñ (ß)
=> INSERT INTO src VALUES ('ss');
=> COMMIT;
```

Querying the `src` table in the default locale returns both ss and sharp s.

```sql
=> SELECT * FROM src;
c1
----
ß
ss
(2 rows)
```

The following query combines pattern-matching predicates to return the results from column `c1`:

```sql
=> SELECT c1, c1 = 'ss' AS equality, c1 LIKE 'ss'
  AS LIKE, c1 ILIKE 'ss' AS ILIKE FROM src;
c1 | equality | LIKE | ILIKE
---|----------|------|-------
ß | False    | False| False
ß | True     | True | False
```
The next query specifies unicode format for c1:

```sql
=> SELECT c1, c1 = U'ß' AS equality,
     c1 LIKE U'ß' AS LIKE,
     c1 ILIKE U'ß' AS ILIKE from src;
```

Now change the locale to German with a strength of 1 (ignore case and accents):

```sql
\locale LDE_S1
=> SELECT c1, c1 = 'ß' AS equality,
     c1 LIKE 'ß' as LIKE, c1 ILIKE 'ß' AS ILIKE from src;
```

This example illustrates binary data types with pattern-matching predicates:

```sql
=> CREATE TABLE t (c BINARY(1));
=> INSERT INTO t values(HEX_TO_BINARY('0x00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFF'));
=> SELECT TO_HEX(c) from t;
```

```sql
select * from t;
```

---

ß | f | f | f
| | | |
ß | t | t | t
(2 rows)

ß | t | t | t
| | | |
ß | f | f | f
(2 rows)

ß | t | t | t
| | | |
ß | t | t | t
(2 rows)

ß | f | f | f
ß | t | t | t
ß | f | f | f
ß | t | t | t
(2 rows)

ß | f | f | f
ß | t | t | t
ß | f | f | f
ß | t | t | t
(2 rows)

(2 rows)
NULL-predicate

Tests for null values.

Syntax

\textit{value_expression} IS [ NOT ] NULL

Parameters

| \textit{value_expression} | A column name, literal, or function. |

Examples

Column name:

\begin{verbatim}
=> SELECT date_key FROM date_dimension WHERE date_key IS NOT NULL;
\end{verbatim}

\begin{verbatim}
date_key
-------
   1
  366
1462
1097
  2
   3
   6
   7
   8
...
\end{verbatim}

Function:

\begin{verbatim}
=> SELECT MAX(household_id) IS NULL FROM customer_dimension;
\end{verbatim}

\begin{verbatim}
?column?
-------
f
(1 row)
\end{verbatim}

Literal:

\begin{verbatim}
=> SELECT 'a' IS NOT NULL;
\end{verbatim}

\begin{verbatim}
?column?
-------
t
\end{verbatim}
Hints

Hints are directives that you embed within a query or directed query. They conform to the following syntax:

/**+ hint-name[, hint-name]... */

Hints are bracketed by comment characters /*+ and */., which can enclose multiple comma-delimited hints. For example:

/**+ DIRECT, LABEL(myLabel) */

Restrictions

When embedding hints, be aware of the following restrictions:

- Do not embed spaces in the comment characters /* and */.
- In general, spaces are allowed before and after the plus (+) character and hint-name; however, some third-party tools do not support spaces embedded inside /*+.

Supported Hints

Vertica supports the following hints:

<table>
<thead>
<tr>
<th>General hints</th>
<th>ALLNODES</th>
<th>Qualifies an EXPLAIN statement to request a query plan that assumes all nodes are active.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENABLE_WITH_CLAUSE MATERIALIZATION</td>
<td>Enables and disables WITH clause materialization for a specific query.</td>
</tr>
<tr>
<td></td>
<td>EARLY_MATERIALIZATION</td>
<td>Specifies early materialization of a table for the current query.</td>
</tr>
<tr>
<td></td>
<td>DIRECT</td>
<td>Specifies to write data directly to disk (ROS); valid only for CREATE [TEMPORARY] TABLE AS,</td>
</tr>
<tr>
<td>Label</td>
<td>INSERT, MERGE, and UPDATE operations.</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td></td>
</tr>
<tr>
<td>LABEL</td>
<td>Labels a query so you can identify it for profiling and debugging.</td>
<td></td>
</tr>
</tbody>
</table>

**Join hints**

<table>
<thead>
<tr>
<th>HINT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNTACTIC_JOIN</td>
<td>Enforces join order and enables other join hints.</td>
</tr>
<tr>
<td>DISTRIBUT</td>
<td>Sets the input operations for a distributed join to broadcast, resegment, local, or filter.</td>
</tr>
<tr>
<td>GBYTYPE</td>
<td>Specifies which algorithm—GROUPBY HASH or GROUPBY PIPELINED—the Vertica query optimizer should use to implement a GROUP BY clause.</td>
</tr>
<tr>
<td>JTYPE</td>
<td>Enforces the join type: merge or hash join.</td>
</tr>
<tr>
<td>UTYPE</td>
<td>Specifies how to combine UNION ALL input.</td>
</tr>
</tbody>
</table>

**Table hints**

<table>
<thead>
<tr>
<th>HINT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJS</td>
<td>Specifies one or more projections to use for a queried table.</td>
</tr>
<tr>
<td>SKIP_PROJS</td>
<td>Specifies which projections to avoid using for a queried table.</td>
</tr>
</tbody>
</table>

**Directed query hints**

<table>
<thead>
<tr>
<th>HINT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORECONST</td>
<td>Maps an input query constant to one or more annotated query constants.</td>
</tr>
<tr>
<td>VERBATIM</td>
<td>Enforces execution of an annotated query exactly as written.</td>
</tr>
</tbody>
</table>

---

**ALLNODES**

Qualifies an `EXPLAIN` statement to request a query plan that assumes all nodes are active. If you omit this hint, the `EXPLAIN` statement produces a query plan that takes into account any nodes that are currently down.

**Syntax**

```
EXPLAIN /*+ ALLNODES */ ...
```
Examples

In the following example, the ALLNODES hint requests a query plan that assumes all nodes are active.

```
QUERY PLAN DESCRIPTION:
---------------------------
Opt Vertica Options
---------------------
PLAN_ALL_NODES_ACTIVE

EXPLAIN /*+ALLNODES*/ select * from Emp_Dimension;
```

Access Path:
+-STORAGE ACCESS for Emp_Dimension [Cost: 125, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
 | Projection: public.Emp_Dimension.b0
 | Execute on: All Nodes

AUTO

Specifies to initially load data into WOS. This hint overrides the table's load method.

The following statements support this hint:

- **INSERT**
- **MERGE**
- **UPDATE**
- **CREATE [TEMPORARY] TABLE AS**

For details, see Choosing a Load Method in the Administrator's Guide.

Syntax

```
statement-name /*+ AUTO */ ...
```
DIRECT

Specifies to bypass memory (WOS) and write table data directly to disk (ROS). This hint overrides the table's load method, by default set to AUTO.

The following statements support this hint:

- DELETE
- INSERT
- MERGE
- UPDATE
- CREATE [TEMPORARY] TABLE AS

For details, see Choosing a Load Method in the Administrator's Guide.

Syntax

```
statement-name /*+ DIRECT */ ...
```

DISTRIBUT

Specifies to the optimizer how to distribute join key data to implement a join.

Syntax

```
...JOIN /*+ DISTRIB(outer-join, inner-join) */
```

Arguments

<table>
<thead>
<tr>
<th>outer-join</th>
<th>inner-join</th>
<th>Specifies how to distribute data on the outer and inner joins:</th>
</tr>
</thead>
<tbody>
<tr>
<td>L (local):</td>
<td></td>
<td>Inner and outer join keys are identically segmented on each</td>
</tr>
<tr>
<td></td>
<td></td>
<td>node, join locally.</td>
</tr>
</tbody>
</table>
### Description

The DISTRIB hint specifies to the optimizer how to distribute join key data in order to implement a join. If a specified distribution is not feasible, the optimizer ignores the hint and throws a warning.

The following requirements apply:

- Queries that include the DISTRIB hint must also include the SYNTACTIC_JOIN hint. Otherwise, the optimizer ignores the DISTRIB hint and throws a warning.

- Join syntax must conform with ANSI SQL-92 join conventions.

### Examples

In the following query, the join is qualified with a DISTRIB hint of `/*+ DISTRIB(L,R)*/`. This hint tells the optimizer to resegment data of join key `stores.store_key` before joining it to the `sales.store_key` data:

```sql
SELECT /*+ SYNTACTIC_JOIN */ sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity, sales.sale_date
FROM (store.storeSales AS sales JOIN /*+DISTRIB(L,R),JTYPE(H)*/ store.store_dimension AS stores ON (sales.store_key = stores.store_key))
WHERE (sales.sale_date = '2014-12-01':date) ORDER BY sales.store_key, sales.sale_date;
```
EARLY_MATERIALIZATION

Specifies early materialization of a table for the current query. A query can include this hint for any number of tables. Typically, the query optimizer delays materialization until late in the query execution process. This hint overrides any choices that the optimizer otherwise would make.

This hint can be useful in cases where late materialization of join inputs precludes other optimizations—for example, pushing aggregation down the joins, or using live aggregate projections. In these cases, qualifying a join input with EARLY_MATERIALIZATION can enable the optimizations.

Syntax

```sql
  table-name [ [AS] alias] /*+ EARLY_MATERIALIZATION */
```

ENABLE_WITH_CLAUSE_MATERIALIZATION

Enables and disables WITH clause materialization for a specific query. Materialization is automatically cleared when the query returns. For more information, see Materialization of WITH Clause in Analyzing Data.

Syntax

```sql
  WITH /*+ENABLE_WITH_CLAUSE_MATERIALIZATION*/ with-query...
```

GBYTYPE

Specifies which algorithm—GROUPBY HASH or GROUPBY PIPELINED—the Vertica query optimizer should use to implement a GROUP BY clause. If both algorithms are valid for this query, the query optimizer chooses the specified algorithm over the algorithm that the query optimizer might otherwise choose in its query plan.
Syntax

```sql
...GROUP BY /*+ GBYTYPE( HASH | PIPE ) */
```

Arguments

<table>
<thead>
<tr>
<th>HASH</th>
<th>PIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the GROUP BY algorithm to use:</td>
<td></td>
</tr>
<tr>
<td>• HASH: GROUPBY HASH algorithm</td>
<td></td>
</tr>
<tr>
<td>• PIPE: GROUPBY PIPELINED algorithm</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Vertica uses the GROUPBY PIPELINED algorithm only if the query and one of its projections comply with GROUP BY PIPELINED requirements. Otherwise, Vertica issues a warning and uses GROUPBY HASH.

For more information about both algorithms, see GROUP BY Implementation Options.

Examples

See Controlling GROUPBY Algorithm Choice in Analyzing Data.

**IGNORECONST**

In a directed query, maps an input query constant to one or more annotated query constants. IGNORECONST lets you create directed queries that support input queries with various conditions. IGNORECONST requires an integer argument. This argument matches constants in input and annotated queries that you want the optimizer to ignore.

For details, see Ignoring Constants in Directed Queries in the Administrator's Guide.

Syntax

```sql
/*+ IGNORECONST(arg) */
```
Examples

In the following example a directed query is created where input and annotated queries set IGNORECONST hints on Employee_city and Employee_position:

```sql
=> SAVE QUERY SELECT Employee_first_name, Employee_last_name FROM EMP_Dimension
   WHERE Employee_city='somewhere'/*+IGNORECONST(1)*/
   AND Employee_position='somejob'/*+IGNORECONST(2)*/;
SAVE QUERY

=> CREATE DIRECTED QUERY CUSTOM 'findEmployees'
   SELECT Employee_first_name, Employee_last_name FROM
   EMP_Dimension /+projs('public.Emp_Dimension_Unseg')/
   WHERE Employee_city='somewhere'/*+IGNORECONST(1)*/
   AND Employee_position='somejob'/*+IGNORECONST(2)*/;
CREATE DIRECTED QUERY
```

IGNORECONST pairs two sets of constants:

- IGNORECONST(1) pairs input and annotated query settings for Employee_city.
- IGNORECONST(2) pairs input and annotated query settings for Employee_position.

JTYPE

Specifies the join algorithm as hash or merge.

Use the JTYPE hint to specify the algorithm the optimizer uses to join table data. If specified algorithm is not feasible, the optimizer ignores the hint and throws a warning.

Syntax

```sql
...JOIN /*+ JTYPE(join-type) */
```

Arguments

<table>
<thead>
<tr>
<th>join-type</th>
<th>One of the following arguments:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H: Hash join</td>
</tr>
<tr>
<td></td>
<td>M: Merge join, valid only if both join columns, otherwise Vertica ignores it and throws a warning.</td>
</tr>
</tbody>
</table>
Note: The optimizer relies upon the query or DDL to verify whether input data is sorted, rather than the actual runtime order of the data.

- FM: Forced merge join. Before performing the merge, the optimizer re-sorts the join inputs. Two restrictions apply:
  - This option is valid only for simple join conditions. For example:
    ```sql
    SELECT /*+ SYNTACTIC_JOIN*/ * FROM x /*+JTYPE(FM)*/ JOIN y ON x.c1 = y.c1;
    ```
  - Join columns must be of the same type and precision or scale. One exception applies: string columns can have different lengths.

Requirements

- Queries that include the JTYPE hint must also include the SYNTACTIC_JOIN hint. Otherwise, the optimizer ignores the JTYPE hint and throws a warning.

- Join syntax must conform with ANSI SQL-92 join conventions.

LABEL

Assigns a label to a statement in order to identify it for profiling and debugging.

LABEL hints are valid in the following statements:

- `DELETE`
- `INSERT`
- `MERGE`
- `SELECT`
- `UPDATE`
- `UNION`: Valid in the UNION's first SELECT statement. Vertica ignores labels in subsequent SELECT statements.

Syntax

```
statement-name /*+ LABEL (label-string) */
```
Arguments

| label-string          | A string that is up to 128 octets long. If enclosed with single quotes, label-string can contain embedded spaces. |

Examples

See Labeling Queries in the Administrator's Guide.

PROJS

Specifies one or more projections to use for a queried table.

Syntax

```sql
...FROM table-name /*+ PROJS( [[database.]schema.]projection[,...] ) */
```

Arguments

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>projection</td>
<td>The projection to use. You can specify a list of comma-delimited projections.</td>
</tr>
</tbody>
</table>

Description

The PROJS hint can specify multiple projections; the optimizer determines which ones are valid and uses the one that is most cost-effective for the queried table. If no hinted projection is valid, the query returns a warning and ignores projection hints.
Examples

The following query includes a PROJS hint that specifies two projections:

```sql
=> EXPLAIN SELECT * FROM Emp_Dimension /*+PROJS('public.Emp_Dimension_Unseg', 'public.Emp_Dimension')*/;
```

The first projection `public.Emp_Dimension_Unseg` does not include all columns in the queried table `Emp_Dimension`, so the optimizer cannot use it. The second projection includes all table columns so the optimizer uses it, as verified by the following query plan:

```
QUERY PLAN DESCRIPTION:
-------------------------
explain SELECT * FROM Emp_Dimension /*+PROJS('public.Emp_Dimension_Unseg', 'public.Emp_Dimension')*/;
Access Path:
+ STORAGE ACCESS for Emp_Dimension [Cost: 125, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
 |   Projection: public.Emp_Dimension_b0
```

**SKIP_PROJS**

Specifies which projections to avoid using for a queried table.

**Syntax**

```sql
...FROM table-name /*+ SKIP_PROJS( [[database.]schema.]projection[,...] ) */
```

**Arguments**

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td>projection</td>
<td>A projection to skip. You can specify a list of comma-delimited projections.</td>
</tr>
</tbody>
</table>
Description

The SKIP_PROJS specifies one or more projections that the optimizer should avoid using. If the SKIP_PROJS hint excludes all available projections that are valid for the query, the optimizer issues a warning and ignores the projection hints.

Examples

In this example, the EXPLAIN output shows that the optimizer uses the projection public.Emp_Dimension_b0 for a given query:

```
QUERY PLAN DESCRIPTION:
-------------------------
EXPLAIN SELECT Employee_last_name, Employee_first_name, Employee_city, Employee_position FROM Emp_Dimension;
Access Path:
  + STORAGE ACCESS for Emp_Dimension [Cost: 59, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
  |   Projection: public.Emp_Dimension_b0
```

You can use the SKIP_PROJS hint to avoid using this projection. If another projection is available that is valid for this query, the optimizer uses it instead:

```
QUERY PLAN DESCRIPTION:
-------------------------
EXPLAIN SELECT Employee_last_name, Employee_first_name, Employee_city, Employee_position FROM Emp_Dimension /*+SKIP_PROJS('public.Emp_Dimension')*/;
Access Path:
  + STORAGE ACCESS for Emp_Dimension [Cost: 152, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
  |   Projection: public.Emp_Dimension_Unseg
```

SYNTACTIC_JOIN

Enforces join order and enables other join hints.

Syntax

```
/*+ SYN[TACTIC]_JOIN */
```
Description

In order to achieve optimal performance, the optimizer often overrides a query's specified join order. By including the SYNTACTIC_JOIN hint, you can ensure that the optimizer enforces the query's join order exactly as specified. One requirement applies: the join syntax must conform with ANSI SQL-92 conventions.

The SYNTACTIC JOIN hint must immediately follow SELECT. If the annotated query includes another hint that must also follow SELECT, such as VERBATIM, combine the two hints together. For example:

```
SELECT /*+ syntactic_join,verbatim */ ...
```

Examples

In the following examples, the optimizer produces different plans for two queries that differ only by including or excluding the SYNTACTIC_JOIN hint.

Excludes SYNTACTIC_JOIN

```
EXPLAIN SELECT sales.store_key, stores.store_name, products.product_description, sales.sales_quantity, sales.sale_date
FROM (store.store_sales sales JOIN products ON sales.product_key=products.product_key)
JOIN store.store_dimension stores ON sales.store_key=stores.store_key
WHERE sales.sale_date='2014-12-01'
order by sales.store_key, sales.sale_date;
```

Access Path:
```
+-SORT [Cost: 14K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
  | Order: sales.store_key ASC, sales.sale_date ASC
  | Execute on: All Nodes
  | -+-JOIN HASH [Cost: 11K, Rows: 100K (NO STATISTICS)] (PATH ID: 2) Outer (RESEGMENT)(LOCAL ROUND ROBIN) Inner (RESEGMENT)
  |     | Join Cond: (sales.product_key = products.product_key)
  |     | Materialize at Input: sales.store_key, sales.product_key, sales.sale_date, sales.sales_quantity
  |     | Execute on: All Nodes
  | -+-JOIN HASH [Cost: 1K, Rows: 100K (NO STATISTICS)] (PATH ID: 3)
  |     | Join Cond: (sales.store_key = stores.store_key)
  |     | Execute on: All Nodes
  | -+-STORAGE ACCESS for sales [Cost: 1K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
  |     | Projection: store.store_sales_b0
  |     | Materialize: sales.store_key
  |     | Filter: (sales.sale_date = '2014-12-01':::date)
  |     | Execute on: All Nodes
  |     | Runtime Filter: (SIP1(HashJoin): sales.store_key)
  | -+-STORAGE ACCESS for stores [Cost: 34, Rows: 250] (PATH ID: 5)
  |     | Projection: store.dimension_DBV10_rep_VMartDesign_node0001
  |     | Materialize: stores.store_key, stores.store_name
  |     | Execute on: All Nodes
```
**Include SYNTACTIC_JOIN**

EXPLAIN SELECT /*+SYNTACTIC_JOIN*/ sales.store_key, stores.store_name, products.product_description, sales.sales_quantity, sales.sale_date
FROM (store.store_sales sales JOIN products ON sales.product_key=products.product_key)
JOIN store.store_dimension stores ON sales.store_key=stores.store_key
WHERE sales.sale_date='2014-12-01' order by sales.store_key, sales.sale_date;

Access Path:
+SORT [Cost: 11K, Rows: 100K (NO STATISTICS)] (PATH ID: 1)
| Order: sales.store_key ASC, sales.sale_date ASC
| Execute on: All Nodes
| +--> JOIN HASH [Cost: 8K, Rows: 100K (NO STATISTICS)] (PATH ID: 2)
| | Join Cond: (sales.store_key = stores.store_key)
| | Execute on: All Nodes
| | +-- Outer -> JOIN HASH [Cost: 7K, Rows: 100K (NO STATISTICS)] (PATH ID: 3) Outer (BROADCAST) (LOCAL ROUND ROBIN)
| | | Join Cond: (sales.product_key = products.product_key)
| | | Execute on: All Nodes
| | | Runtime Filter: (SIP1(HashJoin): sales.store_key)
| | | +-- Outer -> STORAGE ACCESS for sales [Cost: 2K, Rows: 100K (NO STATISTICS)] (PATH ID: 4)
| | | | Projection: store.store_sales_b0
| | | | Materialize: sales.sale_date, sales.store_key, sales.product_key, sales.sales_quantity
| | | | Filter: (sales.sale_date = '2014-12-01':::date)
| | | | Execute on: All Nodes
| | | +-- Inner -> STORAGE ACCESS for products [Cost: 3K, Rows: 60K (NO STATISTICS)] (PATH ID: 5)
| | | | Projection: public.products_b0
| | | | Materialize: products.product_key, products.product_description
| | | | Execute on: All Nodes
| | | +-- Inner -> STORAGE ACCESS for stores [Cost: 34, Rows: 250] (PATH ID: 6)
| | | | Projection: store.store_dimension_DBD_10_rep_VMartDesign_node0001
| | | | Materialize: stores.store_key, stores.store_name
| | | | Execute on: All Nodes

**TRICKLE**

Specifies to load data only into WOS. This hint overrides the table's load method, by default set to AUTO.

The following statements support this hint:

- INSERT
- MERGE
UPDATE

CREATE [TEMPORARY] TABLE AS

For details, see Choosing a Load Method in the Administrator's Guide.

Syntax

statement-name /*+ TRICKLE */ ...

UTYPE

Specifies how to combine UNION ALL input.

Syntax

...UNION ALL /*+ UTYPE(union-type) */

Arguments

<table>
<thead>
<tr>
<th>union-type</th>
<th>One of the following arguments:</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Concatenates UNION ALL input (default).</td>
</tr>
<tr>
<td>M</td>
<td>Merges UNION ALL input in the same sort order as the source query results. This option requires all input from the source queries to use the same sort order; otherwise, Vertica throws a warning and concatenates the UNION ALL input.</td>
</tr>
</tbody>
</table>

Note: The optimizer relies upon the query or DDL to verify whether input data is sorted, rather than the actual runtime order of the data.

Requirements

Queries that include the UTYPE hint must also include the SYNTAXIC_JOIN hint. Otherwise, the optimizer ignores the UTYPE hint and throws a warning.
VERBATIM

Enforces execution of an annotated query exactly as written. VERBATIM directs the optimizer to create a query plan that incorporates all hints in an annotated query. Furthermore, it directs the optimizer not to apply its own plan development processing on query plan components that pertain to those hints.

Usage of this hint varies between optimizer-generated and custom directed queries, as described below.

Syntax

SELECT /*+ VERBATIM */ ...

Requirements

The VERBATIM hint must immediately follow SELECT. If the annotated query includes another hint that must also follow SELECT, such as SYNTACTIC_JOIN, combine the two hints together. For example:

SELECT /*+ syntactic_join,verbatim */ ...

Optimizer-Generated Directed Queries

The optimizer always includes the VERBATIM hint in the annotated queries that it generates for directed queries. For example, given the following CREATE DIRECTED QUERY OPTIMIZER statement:

```sql
=> CREATE DIRECTED QUERY OPTIMIZER getStoreSales SELECT sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity, sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-01' /*+IGNORECONST(1)*/ AND stores.store_name='Store1' /*+IGNORECONST(2)*/ ORDER BY sales.store_key, sales.sale_date;
CREATE DIRECTED QUERY
```

The optimizer generates an annotated query that includes the VERBATIM hint:

```sql
=> SELECT query_name, annotated_query FROM V_CATALOG.DIRECTED_QUERIES WHERE query_name = 'getStoreSales';
-[- RECORD 1 ]--------
query_name    | getStoreSales
```
When the optimizer uses this directed query, it produces a query plan that is equivalent to the query plan that it used when it created the directed query:

```sql
=> ACTIVATE DIRECTED QUERY getStoreSales;
ACTIVATE DIRECTED QUERY

=> EXPLAIN SELECT sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity, sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-04' AND stores.store_name='Store14' ORDER BY sales.store_key, sales.sale_date;

QUERY PLAN DESCRIPTION:
--------------------------

EXPLAIN SELECT sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity, sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-04' AND stores.store_name='Store14' ORDER BY sales.store_key, sales.sale_date;

The following active directed query(query name: getStoreSales) is being executed:
SELECT /*+syntactic_join,verbatim*/ sales.store_key, stores.store_name, sales.product_description, sales.sales_quantity, sales.sale_date FROM store.storesales sales JOIN store.store_dimension stores ON sales.store_key=stores.store_key WHERE sales.sale_date='2014-12-04' AND stores.store_name='Store14' ORDER BY sales.store_key, sales.sale_date;

Access Path:
+-JOIN HASH [Cost: 463, Rows: 622 (NO STATISTICS)] (PATH ID: 2)
    | Join Cond: (sales.store_key = stores.store_key)
    | Materialize at Output: sales.sale_date, sales.sales_quantity, sales.product_description
    | Execute on: All Nodes
    | +- Outer -> STORAGE ACCESS for sales [Cost: 150, Rows: 155K (NO STATISTICS)] (PATH ID: 3)
    |    | Projection: store.storesales_b0
    |    | Materialize: sales.store_key
    |    | Filter: (sales.sale_date = '2014-12-04':date)
    |    | Execute on: All Nodes
    |    | Runtime Filter: (SIP1(HashJoin): stores.store_key)
    | +- Inner -> STORAGE ACCESS for stores [Cost: 35, Rows: 2] (PATH ID: 4)
    |    | Projection: store.store_dimension_DB0_10_rep_VMartDesign_node0001
    |    | Materialize: stores.store_name, stores.store_key
    |    | Filter: (stores.store_name = 'Store14')
    |    | Execute on: All Nodes
```
Custom Directed Queries

The VERBATIM hint is included in a custom directed query only if you explicitly include it in the annotated query that you write for that directed query. When the optimizer uses that directed query, it respects the VERBATIM hint and creates a query plan accordingly.

If you omit the VERBATIM hint when you create a custom directed query, the hint is not stored with the annotated query. When the optimizer uses that directed query, it applies its own plan development processing on the annotated query before it generates a query plan. This query plan might not be equivalent to the query plan that the optimizer would have generated for the Vertica version in which the directed query was created.
# SQL Data Types

The following table summarizes the data types that Vertica supports. It also shows the default placement of null values in projections. The Size column lists uncompressed bytes.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size / bytes</th>
<th>Description</th>
<th>NULL Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Binary</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINARY</td>
<td>1 to 65,000</td>
<td>Fixed-length binary string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>1 to 65,000</td>
<td>Variable-length binary string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>LONG VARBINARY</td>
<td>1 to 32,000,000</td>
<td>Long variable-length binary string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>BYTEA</td>
<td></td>
<td>Synonyms for VARBINARY</td>
<td></td>
</tr>
<tr>
<td>RAW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Boolean</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>1</td>
<td>True or False or NULL</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td><strong>Character / Long</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>1 to 65,000</td>
<td>Fixed-length character string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>1 to 65,000</td>
<td>Variable-length character string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>1 to 32,000,000</td>
<td>Long variable-length character string</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td><strong>Date/Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>8</td>
<td>Represents a month, day, and year</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>Data Type</td>
<td>Size / bytes</td>
<td>Description</td>
<td>NULL Sorting</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>TIME</td>
<td>8</td>
<td>Represents a time of day without timezone</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Synonyms for TIMESTAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME WITH TIMEZONE</td>
<td>8</td>
<td>Represents a time of day with timezone</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>8</td>
<td>Represents a date and time without timezone</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIMEZONE</td>
<td>8</td>
<td>Represents a date and time with timezone</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>8</td>
<td>Measures the difference between two points in time</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>8</td>
<td>Represents an interval measured in days and seconds</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>8</td>
<td>Represents an interval measured in years and months</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>Approximate Numeric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>8</td>
<td>Signed 64-bit IEEE floating point number, requiring 8 bytes of storage</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>FLOAT</td>
<td>8</td>
<td>Signed 64-bit IEEE</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>Data Type</td>
<td>Size / bytes</td>
<td>Description</td>
<td>NULL Sorting</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td>floating point number, requiring 8 bytes of storage</td>
<td></td>
</tr>
<tr>
<td>FLOAT(n)</td>
<td>8</td>
<td>Signed 64-bit IEEE floating point number, requiring 8 bytes of storage</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>FLOAT8</td>
<td>8</td>
<td>Signed 64-bit IEEE floating point number, requiring 8 bytes of storage</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>REAL</td>
<td>8</td>
<td>Signed 64-bit IEEE floating point number, requiring 8 bytes of storage</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td><strong>Exact Numeric</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>8</td>
<td>Signed 64-bit integer, requiring 8 bytes of storage</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>INT</td>
<td>8</td>
<td>Signed 64-bit integer, requiring 8 bytes of storage</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>BIGINT</td>
<td>8</td>
<td>Signed 64-bit integer, requiring 8 bytes of storage</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>INT8</td>
<td>8</td>
<td>Signed 64-bit integer, requiring 8 bytes of storage</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>8</td>
<td>Signed 64-bit integer, requiring 8 bytes of storage</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>TINYINT</td>
<td>8</td>
<td>Signed 64-bit</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>Data Type</td>
<td>Size / bytes</td>
<td>Description</td>
<td>NULL Sorting</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>8+</td>
<td>8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>8+</td>
<td>8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>NUMBER</td>
<td>8+</td>
<td>8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>MONEY</td>
<td>8+</td>
<td>8 bytes for the first 18 digits of precision, plus 8 bytes for each additional 19 digits</td>
<td>NULLS FIRST</td>
</tr>
</tbody>
</table>

**Spatial**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size / bytes</th>
<th>Description</th>
<th>NULL Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>1 to 10,000,000</td>
<td>Coordinates expressed as (x,y) pairs, defined in the Cartesian plane.</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>GEOGRAPHY</td>
<td>1 to 10,000,000</td>
<td>Coordinates expressed in longitude/latitude angular values, measured in degrees</td>
<td>NULLS LAST</td>
</tr>
<tr>
<td>Data Type</td>
<td>Size / bytes</td>
<td>Description</td>
<td>NULL Sorting</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>UUID</td>
<td></td>
<td>Stores universally unique identifiers (UUIDs).</td>
<td>NULLS FIRST</td>
</tr>
<tr>
<td>UUID</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Binary Data Types

Store raw-byte data, such as IP addresses, up to 65000 bytes. Data types BINARY and BINARY VARYING (VARBINARY) are collectively referred to as *binary string types* and the values of binary string types are referred to as *binary strings*. A binary string is a sequence of octets or bytes.

**Note:** BYTEA and RAW are synonyms for VARBINARY.

### Syntax

**BINARY**

\[
\text{BINARY} ( \text{Length} )
\]

**VARBINARY**

\[
\{ \text{VARBINARY} \mid \text{BINARY VARYING} \mid \text{BYTEA} \mid \text{RAW} \} ( \text{max-Length} )
\]

### Parameters

- \( \text{Length} \) | \( \text{max-Length} \)
  Specifies the length of the string or column width, in bytes (octets).

### BINARY and VARBINARY Data Types

BINARY and VARBINARY data types have the following attributes:
• **BINARY**: A fixed-width string of *length* bytes, where the number of bytes is declared as an optional specifier to the type. If length is omitted, the default is 1. Where necessary, values are right-extended to the full width of the column with the zero byte. For example:

```sql
=> SELECT TO_HEX('ab'::BINARY(4));
  to_hex
-------
61620000
```

• **VARBINARY**: A variable-width string up to a length of *max-length* bytes, where the maximum number of bytes is declared as an optional specifier to the type. The default is the default attribute size, which is 80, and the maximum length is 65000 bytes. VARBINARY values are not extended to the full width of the column. For example:

```sql
=> SELECT TO_HEX('ab'::VARBINARY(4));
  to_hex
-------
6162
```

### Input Formats

You can use several formats when working with binary values. The hexadecimal format is generally the most straightforward and is emphasized in Vertica documentation.

Binary values can also be represented in octal format by prefixing the value with a backslash `'\'`.

**Note**: If you use vsql, you must use the escape character (\) when you insert another backslash on input; for example, input `'\141'` as `'\141'`.

You can also input values represented by printable characters. For example, the hexadecimal value `'0x61'` can also be represented by the symbol `'`.  

See [Bulk-Loading Data](#) in the Administrator's Guide.

On input, strings are translated from:

- Hexadecimal representation to a binary value using the function `HEX_TO_BINARY`.
- Bitstring representation to a binary value using the function `BITSTRING_TO_BINARY`.

Both functions take a VARCHAR argument and return a VARBINARY value.
Output Formats

Like the input format, the output format is a hybrid of octal codes and printable ASCII characters. A byte in the range of printable ASCII characters (the range $[0x20, 0x7e]$) is represented by the corresponding ASCII character, with the exception of the backslash ('\'), which is escaped as '\\'. All other byte values are represented by their corresponding octal values. For example, the bytes `{97,92,98,99}`, which in ASCII are `{a,\,b,c}`, are translated to text as 'a\bc'.

Binary Operators and Functions

Binary operators &, ~, |, and # have special behavior for binary data types, as described in Bitwise Operators.

The following aggregate functions are supported for binary data types:

- BIT_AND
- BIT_OR
- BIT_XOR
- MAX
- MIN

BIT_AND, BIT_OR, and BIT_XOR are bit-wise operations that are applied to each non-null value in a group, while MAX and MIN are byte-wise comparisons of binary values.

Like their binary operator counterparts, if the values in a group vary in length, the aggregate functions treat the values as though they are all equal in length by extending shorter values with zero bytes to the full width of the column. For example, given a group containing the values 'ff', null, and 'f', a binary aggregate ignores the null value and treats the value 'f' as 'f0'. Also, like their binary operator counterparts, these aggregate functions operate on VARBINARY types explicitly and operate on BINARY types implicitly through casts. See Data Type Coercion Operators (CAST).
Binary Versus Character Data Types

Binary data types BINARY and VARBINARY are similar to character data types CHAR and VARCHAR, respectively. They differ as follows:

- Binary data types contain byte strings—a sequence of octets or bytes.
- Character data types contain character strings (text).
- The lengths of binary data types are measured in bytes, while character data types are measured in characters.

Examples

The following example shows VARBINARY HEX_TO_BINARY (VARCHAR) and VARCHAR TO_HEX (VARBINARY) usage.

Table \( t \) and its projection are created with binary columns:

\[
\begin{align*}
&=> \text{CREATE TABLE} \ t (c \ \text{BINARY}(1)); \\
&=> \text{CREATE PROJECTION} \ t_\ p (c) \ \text{AS SELECT} \ c \ \text{FROM} \ t;
\end{align*}
\]

Insert minimum byte and maximum byte values:

\[
\begin{align*}
&=> \text{INSERT INTO} \ t \ \text{VALUES} (\text{HEX_TO_BINARY}('0x00')); \\
&=> \text{INSERT INTO} \ t \ \text{VALUES} (\text{HEX_TO_BINARY}('0xFF'));
\end{align*}
\]

Binary values can then be formatted in hex on output using the TO_HEX function:

\[
\begin{align*}
&=> \text{SELECT} \ \text{TO_HEX}(c) \ \text{FROM} \ t; \\
&\quad \text{to_hex} \\
&\quad \quad \text{00} \\
&\quad \quad \text{ff} \\
&\quad (2 \ \text{rows})
\end{align*}
\]

The BIT_AND, BIT_OR, and BIT_XOR functions are interesting when operating on a group of values. For example, create a sample table and projections with binary columns:

The example that follows uses table \( t \) with a single column of VARBINARY data type:

\[
\begin{align*}
&=> \text{CREATE TABLE} \ t (c \ \text{VARBINARY}(2)); \\
&=> \text{INSERT INTO} \ t \ \text{VALUES} (\text{HEX_TO_BINARY}('0xFF00')); \\
&=> \text{INSERT INTO} \ t \ \text{VALUES} (\text{HEX_TO_BINARY}('0xFFFF')); \\
&=> \text{INSERT INTO} \ t \ \text{VALUES} (\text{HEX_TO_BINARY}('0xF00F'));
\end{align*}
\]

Query table \( t \) to see column \( c \) output:
Now issue the bitwise AND operation. Because these are aggregate functions, an implicit GROUP BY operation is performed on results using (ff00&ffff&f00f):

```sql
=> SELECT TO_HEX(BIT_AND(c)) FROM t;
  TO_HEX
---------
  f000
(1 row)
```

Issue the bitwise OR operation on (ff00 | ffff | f00f):

```sql
=> SELECT TO_HEX(BIT_OR(c)) FROM t;
  TO_HEX
---------
  ffff
(1 row)
```

Issue the bitwise XOR operation on (ff00#ffff#f00f):

```sql
=> SELECT TO_HEX(BIT_XOR(c)) FROM t;
  TO_HEX
---------
  f0f0
(1 row)
```

**Boolean Data Type**

Vertica provides the standard SQL type BOOLEAN, which has two states: true and false. The third state in SQL boolean logic is unknown, which is represented by the NULL value.

**Syntax**

```sql
BOOLEAN
```

**Parameters**

Valid literal data values for input are:
<table>
<thead>
<tr>
<th>TRUE</th>
<th>'t'</th>
<th>'true'</th>
<th>'y'</th>
<th>'yes'</th>
<th>'1'</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>'f'</td>
<td>'false'</td>
<td>'n'</td>
<td>'no'</td>
<td>'0'</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes

- Do not confuse the BOOLEAN data type with Boolean Operators or the Boolean-Predicate.
- The keywords TRUE and FALSE are preferred and are SQL-compliant.
- A Boolean value of NULL appears last (largest) in ascending order.
- All other values must be enclosed in single quotes.
- Boolean values are output using the letters t and f.

See Also

- NULL Value
- Data Type Coercion Chart

Character Data Types

Stores strings of letters, numbers, and symbols. Data types CHARACTER (CHAR) and CHARACTER VARYING (VARCHAR) are collectively referred to as character string types, and the values of character string types are known as character strings.

Character data can be stored as fixed-length or variable-length strings. Fixed-length strings are right-extended with spaces on output; variable-length strings are not extended.

String literals in SQL statements must be enclosed in single quotes.

Syntax

CHAR

{ CHAR | CHARACTER } ( octet-length )
**VARCHAR**

```
{ VARCHAR | CHARACTER VARYING } (octet-length)
```

### Parameters

<table>
<thead>
<tr>
<th><strong>octet-length</strong></th>
<th>Specifies the length of the string or column width, declared in bytes (octets).</th>
</tr>
</thead>
</table>

### CHAR Versus VARCHAR Data Types

The following differences apply to CHAR and VARCHAR data:

- **CHAR** is conceptually a fixed-length, blank-padded string. Trailing blanks (spaces) are removed on input, and are restored on output. The default length is 1, and the maximum length is 65000 octets (bytes).

- **VARCHAR** is a variable-length character data type. The default length is 80, and the maximum length is 65000 octets. For string values longer than 65000, use [Long Data Types](https://www.vertica.com/docs/9.x/v19715.jhtml). Values can include trailing spaces.

Normally, you use VARCHAR for all of string data. Use CHAR when you need fixed-width string output. For example, you can use CHAR columns for data to be transferred to a legacy system that requires fixed-width strings.

### Setting Maximum Length

When you define character columns, specify the maximum size of any string to be stored in a column. For example, to store strings up to 24 octets in length, use one of the following definitions:

```
CHAR(24) /* fixed-length */
VARCHAR(24) /* variable-length */
```

The maximum length parameter for VARCHAR and CHAR data types refers to the number of octets that can be stored in that field, not the number of characters (Unicode code points). When using multibyte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 octets per character, depending on the data. If the data loaded into a VARCHAR/CHAR column exceeds the specified maximum size for that column, data is truncated on UTF-8 character
boundaries to fit within the specified size. See COPY.

**Note:** Remember to include the extra octets required for multibyte characters in the column-width declaration, keeping in mind the 65000 octet column-width limit.

Due to compression in Vertica, the cost of overestimating the length of these fields is incurred primarily at load time and during sorts.

### NULL Versus NUL

NULL and NUL differ as follows:

- **NUL** represents a character whose ASCII/Unicode code is 0, sometimes qualified "ASCII NUL".

- **NULL** means no value, and is true of a field (column) or constant, not of a character.

CHAR, LONG VARCHAR, and VARCHAR string data types accept ASCII NUL values.

NULL appears last (largest) in ascending order. See also **GROUP BY Clause** for additional information about NULL ordering.

The following example casts the input string containing NUL values to VARCHAR:

```sql
=> SELECT 'vert\0ica'::CHARACTER VARYING AS VARCHAR;

<table>
<thead>
<tr>
<th>CHARACTER VARYING</th>
</tr>
</thead>
<tbody>
<tr>
<td>vert\0ica</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

The result contains 9 characters:

```sql
=> SELECT LENGTH('vert\0ica'::CHARACTER VARYING);

<table>
<thead>
<tr>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

If you use an **extended string literal**, the length is 8 characters:

```sql
=> SELECT E'vert\0ica'::CHARACTER VARYING AS VARCHAR;

<table>
<thead>
<tr>
<th>CHARACTER VARYING</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertica</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

=> SELECT LENGTH(E'vert\0ica'::CHARACTER VARYING);

<table>
<thead>
<tr>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
```
Date/Time Data Types

Vertica supports the full set of SQL date and time data types. In most cases, a combination of DATE, DATETIME, SMALLDATETIME, TIME, TIMESTAMP WITHOUT TIME ZONE, and TIMESTAMP WITH TIME ZONE, and INTERVAL provides a complete range of date/time functionality required by any application.

In compliance with the SQL standard, Vertica also supports the TIME WITH TIME ZONE data type.

The following table lists the characteristics about the date/time data types. All these data types have a size of 8 bytes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Low Value</th>
<th>High Value</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>Dates only (no time of day)</td>
<td>~ 25e+15 BC</td>
<td>~ 25e+15 AD</td>
<td>1 day</td>
</tr>
<tr>
<td>TIME [(p)]</td>
<td>Time of day only (no date)</td>
<td>00:00:00.00</td>
<td>23:59:60.999999</td>
<td>1 μs</td>
</tr>
<tr>
<td>TIMETZ [(p)]</td>
<td>Time of day only, with time zone</td>
<td>00:00:00.00+14</td>
<td>23:59:59.999999-14</td>
<td>1 μs</td>
</tr>
<tr>
<td>TIMESTAMP [(p)]</td>
<td>Both date and time, without time zone</td>
<td>290279-12-22 19:59:05.224194 BC</td>
<td>294277-01-09 04:00:54.775806 AD</td>
<td>1 μs</td>
</tr>
<tr>
<td>TIMESTAMPTZ [(p)]</td>
<td>Both date and time, with time zone</td>
<td>290279-12-22 19:59:05.224194 BC UTC</td>
<td>294277-01-09 04:00:54.775806 AD UTC</td>
<td>1 μs</td>
</tr>
<tr>
<td>INTERVAL [(p)]DAY TO SECOND</td>
<td>Time intervals</td>
<td>-106751991 days 04:00:54.775807</td>
<td>+106751991 days 04:00:54.775807</td>
<td>1 μs</td>
</tr>
<tr>
<td>INTERVAL [(p)]YEAR TO MONTH</td>
<td>Time intervals</td>
<td>~ -768e15 yrs</td>
<td>~ 768e15 yrs</td>
<td>1 month</td>
</tr>
</tbody>
</table>

Time Zone Abbreviations for Input

Vertica recognizes the files in /opt/vertica/share/timezonesets as date/time input values and defines the default list of strings accepted in the AT TIME ZONE zone parameter. The names are not necessarily used for date/time output—output is driven by the official time zone abbreviations associated with the currently selected time zone parameter setting.
Notes

- In Vertica, TIME ZONE is a synonym for TIMEZONE.

- Vertica uses Julian dates for all date/time calculations, which can correctly predict and calculate any date more recent than 4713 BC to far into the future, based on the assumption that the average length of the year is 365.2425 days.

- All date/time types are stored in eight bytes.

- A date/time value of NULL appears first (smallest) in ascending order.

- All the date/time data types accept the special literal value NOW to specify the current date and time. For example:

  ```
  => SELECT TIMESTAMP 'NOW';
  
  ----------------------
  2012-03-13 11:42:22.766989
  (1 row)
  ```

- In Vertica, the INTERVAL data type is SQL:2008 compliant and allows modifiers, called interval qualifiers, that divide the INTERVAL type into two primary subtypes, DAY TO SECOND (the default) and YEAR TO MONTH. You use the SET INTERVALSTYLE command to change the intervalstyle run-time parameter for the current session.

  Intervals are represented internally as some number of microseconds and printed as up to 60 seconds, 60 minutes, 24 hours, 30 days, 12 months, and as many years as necessary. Fields can be positive or negative.

See Also

- TZ Environment Variable
- Using Time Zones With Vertica
- Sources for Time Zone and Daylight Saving Time Data
DATE

Consists of a month, day, and year.

Syntax

DATE

Parameters/Limits

<table>
<thead>
<tr>
<th>Low Value</th>
<th>High Value</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 25e+15 BC</td>
<td>~ 25e+15 AD</td>
<td>1 DAY</td>
</tr>
</tbody>
</table>

See SET DATESTYLE for information about ordering.

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 8, 1999</td>
<td>Unambiguous in any datestyle input mode</td>
</tr>
<tr>
<td>1999-01-08</td>
<td>ISO 8601; January 8 in any mode (recommended format)</td>
</tr>
<tr>
<td>1/8/1999</td>
<td>January 8 in MDY mode; August 1 in DMY mode</td>
</tr>
<tr>
<td>1/18/1999</td>
<td>January 18 in MDY mode; rejected in other modes</td>
</tr>
<tr>
<td>01/02/03</td>
<td>January 2, 2003 in MDY mode</td>
</tr>
<tr>
<td></td>
<td>February 1, 2003 in DMY mode</td>
</tr>
<tr>
<td></td>
<td>February 3, 2001 in YMD mode</td>
</tr>
<tr>
<td>1999-Jan-08</td>
<td>January 8 in any mode</td>
</tr>
<tr>
<td>Jan-08-1999</td>
<td>January 8 in any mode</td>
</tr>
<tr>
<td>08-Jan-1999</td>
<td>January 8 in any mode</td>
</tr>
<tr>
<td>99-Jan-08</td>
<td>January 8 in YMD mode, else error</td>
</tr>
<tr>
<td>08-Jan-99</td>
<td>January 8, except error in YMD mode</td>
</tr>
<tr>
<td>Jan-08-99</td>
<td>January 8, except error in YMD mode</td>
</tr>
<tr>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>19990108</td>
<td>ISO 8601; January 8, 1999 in any mode</td>
</tr>
<tr>
<td>9990108</td>
<td>ISO 8601; January 8, 1999 in any mode</td>
</tr>
<tr>
<td>1999.008</td>
<td>Year and day of year</td>
</tr>
<tr>
<td>J2451187</td>
<td>Julian day</td>
</tr>
<tr>
<td>January 8, 99 BC</td>
<td>Year 99 before the Common Era</td>
</tr>
</tbody>
</table>

**DATETIME**

DATETIME is an alias for **TIMESTAMP/TIMESTAMPZ**.

**INTERVAL**

Measures the difference between two points in time. Intervals can be positive or negative. The INTERVAL data type is divided into two major subtypes:

- **Year-month**: Span of years and months
- **Day-time**: Span of days, hours, minutes, seconds, and fractional seconds

**Syntax**

```
INTERVAL [ (p) ] 'interval-literal' [ interval-qualifier ]
```

**Parameters**

<table>
<thead>
<tr>
<th>p</th>
<th>Specifies precision of the seconds field, where p is an integer between 0 - 6. For details, see Specifying Interval Precision. Default: 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interval-literal</code></td>
<td>A character string that expresses an interval, conforms to this format: <code>[ - ] { quantity subtype-unit }[...][ago]</code></td>
</tr>
</tbody>
</table>

For details, see Interval Literal.

interval-qualifier

Optionally specifies how to interpret and format an interval literal for output, and, optionally, sets precision. If omitted, the default is DAY TO SECOND(6). For details, see Interval Qualifier.

## Limits

<table>
<thead>
<tr>
<th>Name</th>
<th>Low Value</th>
<th>High Value</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERVAL [(p)] DAY TO SECOND</td>
<td>-106751991 days 04:00:54.775807</td>
<td>+/ -106751991 days 04:00:54.775807</td>
<td>1 microsecond</td>
</tr>
<tr>
<td>INTERVAL [(p)] YEAR TO MONTH</td>
<td>~/-768e15 yrs</td>
<td>~ 768e15 yrs</td>
<td>1 month</td>
</tr>
</tbody>
</table>

## Setting Interval Unit Display

You can control how Vertica displays interval units in a SELECT INTERVAL query, with SET INTERVALSTYLE and SET DATESTYLE. These statements only affect interval output format.

Important: DATESTYLE settings supersede INTERVALSTYLE. If DATESTYLE is set to SQL, interval unit display always conforms to the SQL:2008 standard, which omits interval unit display. If DATESTYLE is set to ISO, you can use SET INTERVALSTYLE to omit or display interval unit display, as described below.

## Omitting Interval Units

To omit interval units from the output, set INTERVALSTYLE to PLAIN. This is the default setting, which conforms with the SQL:2008 standard:

```sql
=> SET INTERVALSTYLE TO PLAIN;
SET
=> SELECT INTERVAL '3 2';
?column?
----------
3 02:00
```
When `INTERVALSTYLE` is set to `PLAIN`, units are omitted from the output, even if the query specifies input units:

```
=> SELECT INTERVAL '3 days 2 hours';
?column?
-----------
3 02:00
```

If `DATESTYLE` is set to `SQL`, Vertica conforms with SQL:2008 standard and always omits interval units from output:

```
=> SET DATESTYLE TO SQL;
SET
=> SET INTERVALSTYLE TO UNITS;
SET
=> SELECT INTERVAL '3 2';
?column?
-----------
3 02:00
```

Displaying Interval Units

To enable display of interval units, `DATESTYLE` must be set to `ISO`. You can then display interval units by setting `INTERVALSTYLE` to `UNITS`:

```
=> SET DATESTYLE TO ISO;
SET
=> SET INTERVALSTYLE TO UNITS;
SET
=> SELECT INTERVAL '3 2';
?column?
-----------
3 days 2 hours
```

Checking `INTERVALSTYLE` and `DATESTYLE` Settings

Use `SHOW` statements to check `INTERVALSTYLE` and `DATESTYLE` settings:

```
=> SHOW INTERVALSTYLE;
 name | setting
-----------
 intervalstyle | units
=> SHOW DATESTYLE;
 name | setting
-----------
datestyle | ISO, MDY
```
Specifying Interval Input

Interval values are expressed through interval literals. An interval literal is composed of one or more interval fields, where each field represents a span of days and time, or years and months, as follows:

\[- \{ \text{quantity}\ \text{subtype-unit}\ } [...\] [AGO]

Using Subtype Units

Subtype units are optional for day-time intervals; they must be specified for year-month intervals.

For example, the first statement below implicitly specifies days and time; the second statement explicitly identifies day and time units. Both statements return the same result:

```
=> SET INTERVALSTYLE TO UNITS;
=> SELECT INTERVAL '1 12:59:10:05';
 ?column?
-------------
 1 day 12:59:10.005
(1 row)
```

```
=> SELECT INTERVAL '1 day 12 hours 59 min 10 sec 5 milliseconds';
 ?column?
-------------
 1 day 12:59:10.005
(1 row)
```

The following two statements add 28 days and 4 weeks to the current date, respectively. The intervals in both cases are equal and the statements return the same result. However, in the first statement, the interval literal omits the subtype (implicitly days); in the second statement, the interval literal must include the subtype unit weeks:

```
=> SELECT CURRENT_DATE;
 ?column?
------------
 2016-08-15
(1 row)
```

```
=> SELECT CURRENT_DATE + INTERVAL '28';
 ?column?
------------
 2016-09-12 00:00:00
(1 row)
```

```
dbadmin=> SELECT CURRENT_DATE + INTERVAL '4 weeks';
 ?column?
------------
 2016-09-12 00:00:00
```
An interval literal can include day-time and year-month fields. For example, the following statement adds an interval of 4 years, 4 weeks, 4 days and 14 hours to the current date. The years and weeks fields must include subtype units; the days and hours fields omit them:

```
> SELECT CURRENT_DATE + INTERVAL '4 years 4 weeks 4 14';
```

```
2020-09-15 14:00:00
(1 row)
```

### Omitting Subtype Units

You can specify quantities of days, hours, minutes, and seconds without specifying units. Vertica recognizes colons in interval literals as part of the timestamp:

```
=> SELECT INTERVAL '1 4 5 6';
```

```
1 day 04:05:06
```

```
=> SELECT INTERVAL '1 4:5:6';
```

```
1 day 04:05:06
```

```
=> SELECT INTERVAL '1 day 4 hour 5 min 6 sec';
```

```
1 day 04:05:06
```

If Vertica cannot determine the units, it applies the quantity to any missing units based on the interval qualifier. In the next two examples, Vertica uses the default interval qualifier (DAY TO SECOND(6)) and assigns the trailing 1 to days, since it has already processed hours, minutes, and seconds in the output:

```
=> SELECT INTERVAL '4:5:6 1';
```

```
1 day 04:05:06
```

```
=> SELECT INTERVAL '1 4:5:6';
```

```
1 day 04:05:06
```

In the next two examples, Vertica recognizes 4 : 5 as hours:minutes. The remaining values in the interval literal are assigned to the missing units: 1 is assigned to days and 2 is assigned to seconds.
SELECT INTERVAL '4:5 1 2';
?column?
-----------
1 day 04:05:02
=> SELECT INTERVAL '1 4:5 2';
?column?
-----------
1 day 04:05:02

Specifying the interval qualifier can change how Vertica interprets 4:5:

=> SELECT INTERVAL '4:5' MINUTE TO SECOND;
?column?
-----------
00:04:05

### Controlling Interval Format

**Interval qualifiers** specify a range of options that Vertica uses to interpret and format an interval literal. The interval qualifier can also specify precision. Each interval qualifier is composed of one or two units:

```
unit[p] [ TO unit[p] ]
```

where:

- **unit** specifies a day-time or year-month subtype.
- **p** specifies precision, an integer between 0 and 6. In general, precision only applies to SECOND units. The default precision for SECOND is 6. For details, see [Specifying Interval Precision](#).

If an interval omits an interval qualifier, Vertica uses the default **DAY TO SECOND(6)**.

### Interval Qualifier Categories

Interval qualifiers belong to one of the following categories:

- **Year-month**: Span of years and months
- **Day-time**: Span of days, hours, minutes, seconds, and fractional seconds

**Year-Month**

Vertica supports two year-month subtypes: **YEAR** and **MONTH**.

---

**Note:** Vertica Documentation SQL Reference Manual 9.0.x
In the following example, YEAR TO MONTH qualifies the interval literal 1 2 to indicate a span of 1 year and two months:

```sql
=> SELECT interval '1 2' YEAR TO MONTH;

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year 2 months</td>
</tr>
</tbody>
</table>

(1 row)
```

If you omit the qualifier, Vertica uses the default interval qualifier DAY TO SECOND and returns a different result:

```sql
=> SELECT interval '1 2';

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 day 2 hours</td>
</tr>
</tbody>
</table>

(1 row)
```

The following example uses the interval qualifier YEAR. In this case, Vertica extracts only the year from the interval literal 1y 10m:

```sql
=> SELECT INTERVAL '1y 10m' YEAR;

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year</td>
</tr>
</tbody>
</table>

(1 row)
```

In the next example, the interval qualifier MONTH converts the same interval literal to months:

```sql
=> SELECT INTERVAL '1y 10m' MONTH;

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 months</td>
</tr>
</tbody>
</table>

(1 row)
```

**Day-Time**

Vertica supports four day-time subtypes: DAY, HOUR, MINUTE, and SECOND.

In the following example, the interval qualifier DAY TO SECOND(4) qualifies the interval literal 1h 3m 6s 5msecs 57us. The qualifier also sets precision on seconds to 4:

```sql
=> SELECT INTERVAL '1h 3m 6s 5msecs 57us' DAY TO SECOND(4);

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:03:06.0051</td>
</tr>
</tbody>
</table>

(1 row)
```

If no interval qualifier is specified, Vertica uses the default subtype DAY TO SECOND(6), regardless of how you specify the interval literal. For example, as an extension to SQL:2008, both of the following commands return 910 days:

```sql
=> SELECT INTERVAL '1h 3m 6s 5msecs 57us' DAY TO SECOND(6);

<table>
<thead>
<tr>
<th>column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:03:06.0051</td>
</tr>
</tbody>
</table>

(1 row)
```
An interval qualifier can extract other values from the input parameters. For example, the following command extracts the HOUR value from the interval literal 3 days 2 hours:

```sql
=> SELECT INTERVAL '3 days 2 hours' HOUR;
?column?
          -------
          74 hours
```

The primary day/time (DAY TO SECOND) and year/month (YEAR TO MONTH) subtype ranges can be restricted to more specific range of types by an interval qualifier. For example, HOUR TO MINUTE is a limited form of day/time interval, which can be used to express time zone offsets.

```sql
=> SELECT INTERVAL '1 3' HOUR to MINUTE;
?column?
          -------
          01:03
```

hh:mm:ss and hh:mm formats are used only when at least two of the fields specified in the interval qualifier are non-zero and there are no more than 23 hours or 59 minutes:

```sql
=> SELECT INTERVAL '2 days 12 hours 15 mins' DAY TO MINUTE;
?column?
          -------
          2 days 12:15
=> SELECT INTERVAL '15 mins 20 sec' MINUTE TO SECOND;
?column?
          -------
          00:15:20
=> SELECT INTERVAL '1 hour 15 mins 20 sec' MINUTE TO SECOND;
?column?
          -------
          75 mins 20 secs
```

### Specifying Interval Precision

In general, interval precision only applies to seconds. If no precision is explicitly specified, Vertica rounds precision to a maximum of six decimal places. For example:

```sql
=> SELECT INTERVAL '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND;
?column?
          -------
```
Vertica lets you specify interval precision in two ways:

- After the INTERVAL keyword

- After the SECOND unit of an interval qualifier, one of the following:
  - DAY TO SECOND
  - HOUR TO SECOND
  - MINUTE TO SECOND
  - SECOND

For example, the following statements use both methods to set precision, and return identical results:

```sql
=> SELECT INTERVAL(4) '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND;
?column?
-----------
02:04:03.7094
(1 row)

=> SELECT INTERVAL '2 hours 4 minutes 3.709384766 seconds' DAY TO SECOND(4);
?column?
-----------
02:04:03.7094
(1 row)
```

If the same statement specifies precision more than once, Vertica uses the lesser precision. For example, the following statement specifies precision twice: the INTERVAL keyword specifies precision of 1, while the interval qualifier SECOND specifies precision of 2. Vertica uses the lesser precision of 1:

```sql
=> SELECT INTERVAL(1) '1.2467' SECOND(2);
?column?
--------
1.2 secs
```

Setting Precision on Interval Table Columns

If you create a table with an interval column, the following restrictions apply to the column definition:
You can set precision on the INTERVAL keyword only if you omit specifying an interval qualifier. If you try to set precision on the INTERVAL keyword and include an interval qualifier, Vertica returns an error.

You can set precision only on the last unit of an interval qualifier. For example:

```sql
CREATE TABLE public.testint2
(
    i INTERVAL HOUR TO SECOND(3)
);
```

If you specify precision on another unit, Vertica discards it when it saves the table definition.

### Fractional Seconds in Interval Units

Vertica supports intervals in milliseconds (hh:mm:ss:ms), where 01:02:03:25 represents 1 hour, 2 minutes, 3 seconds, and 025 milliseconds. Milliseconds are converted to fractional seconds as in the following example, which returns 1 day, 2 hours, 3 minutes, 4 seconds, and 25.5 milliseconds:

```sql
=> SELECT INTERVAL '1 02:03:04:25.5';
  ?column?
  1 day 02:03:04.0255
```

Vertica allows fractional minutes. The fractional minutes are rounded into seconds:

```sql
=> SELECT INTERVAL '10.5 minutes';
  ?column?
  00:10:30
=> select interval '10.659 minutes';
  ?column?
  00:10:39.54
=> select interval '10.3333333333333 minutes';
  ?column?
  00:10:20
```

### Considerations

- An INTERVAL can include only the subset of units that you need; however, year/month intervals represent calendar years and months with no fixed number of days, so year/month interval values cannot include days, hours, minutes. When year/month values are specified for day/time intervals, the intervals extension assumes 30 days per month and 365 days per
year. Since the length of a given month or year varies, day/time intervals are never output as months or years, only as days, hours, minutes, and so on.

- Day/time and year/month intervals are logically independent and cannot be combined with or compared to each other. In the following example, an interval.literal that contains DAYS cannot be combined with the YEAR TO MONTH type:

  ```sql
  => SELECT INTERVAL '1 2 3' YEAR TO MONTH;
  ERROR 3679: Invalid input syntax for interval year to month: "1 2 3"
  ```

- Vertica accepts intervals up to \(2^{63} - 1\) microseconds or months (about 18 digits).

- ``INTERVAL YEAR TO MONTH`` can be used in an analytic RANGE window when the ORDER BY column type is TIMESTAMP/TIMESTAMP WITH TIMEZONE, or DATE. Using TIME/TIME WITH TIMEZONE are not supported.

- You can use ``INTERVAL DAY TO SECOND`` when the ORDER BY column type is TIMESTAMP/TIMESTAMP WITH TIMEZONE, DATE, and TIME/TIME WITH TIMEZONE.

### Examples

Examples in this section assume that INTERVALSTYLE is set to PLAIN, so results omit subtype units. Interval values that omit an interval qualifier use the default to DAY TO SECOND(6).

<table>
<thead>
<tr>
<th>Query</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT INTERVAL '00:2500:00';</td>
<td>17:40</td>
</tr>
<tr>
<td>SELECT INTERVAL '2500' MINUTE TO SECOND;</td>
<td>2500</td>
</tr>
<tr>
<td>SELECT INTERVAL '2500' MINUTE;</td>
<td>2500</td>
</tr>
<tr>
<td>SELECT INTERVAL '28 days 3 hours' HOUR TO SECOND;</td>
<td>675:00</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3 hours';</td>
<td>28:03:00</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3 hours 1.234567';</td>
<td>28:03:01:14.074</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3 hours 1.234567 sec';</td>
<td>28:03:00:01.235</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3.3 hours' HOUR TO SECOND;</td>
<td>675:18</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3.35 hours' HOUR TO SECOND;</td>
<td>675:21</td>
</tr>
<tr>
<td>SELECT INTERVAL(3) '28 days 3.37 hours' HOUR TO SECOND;</td>
<td>675:22:12</td>
</tr>
<tr>
<td>SELECT INTERVAL '1.234567 days' HOUR TO SECOND;</td>
<td>02:37:46.5888</td>
</tr>
<tr>
<td>SELECT INTERVAL '1.23456789 days' HOUR TO SECOND;</td>
<td>29:37:46.665696</td>
</tr>
</tbody>
</table>
### Processing Signed Intervals

In the SQL:2008 standard, a minus sign before an interval-literal or as the first character of the interval-literal negates the entire literal, not just the first component. In Vertica, a leading minus sign negates the entire interval, not just the first component. The following commands both return the same value:

```sql
=> SELECT INTERVAL '-1 month - 1 second';
?column?
------------
-29 days 23:59:59

=> SELECT INTERVAL '-1 month - 1 second';
?column?
------------
-29 days 23:59:59
```

Use one of the following commands instead to return the intended result:

```sql
=> SELECT INTERVAL '-1 month 1 second';
?column?
```
Two negatives together return a positive:

```sql
=> SELECT INTERVAL '-1 month - 1 second';
?column?
----------
29 days 23:59:59
=> SELECT INTERVAL '-1 month 1 second';
?column?
----------
30 days 1 sec
```

You can use the year-month syntax with no spaces. Vertica allows the input of negative months but requires two negatives when paired with years.

```sql
=> SELECT INTERVAL '3-3' YEAR TO MONTH;
?column?
----------
3 years 3 months
=> SELECT INTERVAL '3-3' YEAR TO MONTH;
?column?
----------
2 years 9 months
```

When the interval-literal looks like a year/month type, but the type is day/second, or vice versa, Vertica reads the interval-literal from left to right, where number-number is years-months, and number <space> <signed number> is whatever the units specify. Vertica processes the following command as \((-\) 1 year 1 month = \((-\) 365 + 30 = \(-395\) days:

```sql
=> SELECT INTERVAL '-1-1' DAY TO HOUR;
?column?
----------
-395 days
```

If you insert a space in the interval-literal, Vertica processes it based on the subtype DAY TO HOUR: \((-\) 1 day – 1 hour = \((-\) 24 – 1 = \(-23\) hours:

```sql
=> SELECT INTERVAL '-1 -1' DAY TO HOUR;
?column?
----------
-23 hours
```

Two negatives together returns a positive, so Vertica processes the following command as \((-\) 1 year – 1 month = \((-\) 365 – 30 = \(-335\) days:
SELECT INTERVAL '-1-1' DAY TO HOUR;
  ?column?
 ------------
 -335 days

If you omit the value after the hyphen, Vertica assumes 0 months and processes the following command as 1 year 0 month –1 day = 365 + 0 – 1 = –364 days:

=> SELECT INTERVAL '1- -1' DAY TO HOUR;
  ?column?
 ------------
 364 days

Casting with Intervals

You can use CAST to convert strings to intervals, and vice versa.

String to Interval

You cast a string to an interval as follows:

CAST( [ INTERVAL[(p)] ] [-] ] interval-literal AS INTERVAL[(p)] interval-qualifier )

For example:

=> SELECT CAST('3700 sec' AS INTERVAL);
  ?column?
 ------------
 01:01:40

You can cast intervals within day-time or the year-month subtypes but not between them:

=> SELECT CAST(INTERVAL '4440' MINUTE as INTERVAL);
  ?column?
 ------------
 3 days 2 hours
=> SELECT CAST(INTERVAL '-01:15' as INTERVAL MINUTE);
  ?column?
 ------------
-75 mins

Interval to String

You cast an interval to a string as follows:

CAST( (SELECT interval ) AS VARCHAR[(n)] )

For example:
Operations with Intervals

If you divide an interval by an interval, you get a FLOAT:

```sql
=> SELECT INTERVAL '28 days 3 hours' HOUR(4) / INTERVAL '27 days 3 hours' HOUR(4);
?column?
-------------
1.036866359447
```

An INTERVAL divided by FLOAT returns an INTERVAL:

```sql
=> SELECT INTERVAL '3' MINUTE / 1.5;
?column?
-------------
2 mins
```

INTERVAL MODULO (remainder) INTERVAL returns an INTERVAL:

```sql
=> SELECT INTERVAL '28 days 3 hours' HOUR % INTERVAL '27 days 3 hours' HOUR;
?column?
-------------
24 hours
```

If you add INTERVAL and TIME, the result is TIME, modulo 24 hours:

```sql
=> SELECT INTERVAL '1' HOUR + TIME '1:30';
?column?
-------------
02:30:00
```

SMALLDATETIME

SMALLDATETIME is an alias for TIMESTAMP/TIMESTAMPTZ.
TIME/TIMETZ

Stores the specified time of day. TIMETZ is the same as TIME WITH TIME ZONE: both data types store the UTC offset of the specified time.

Syntax

TIME

TIME [ (p) ] [ { WITHOUT | WITH } TIME ZONE ] 'input-string' [ AT TIME ZONE zone ]

TIMETZ

TIME [ (p) ] 'input-string' [ AT TIME ZONE zone ]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Optional precision value that specifies the number of fractional digits retained in the seconds field, an integer value between 0 and 6. If you omit specifying precision, Vertica returns up to 6 fractional digits.</td>
</tr>
<tr>
<td>WITHOUT TIME ZONE</td>
<td>Specifies whether to include a time zone with the stored value:</td>
</tr>
<tr>
<td>WITH TIME ZONE</td>
<td>- WITHOUT TIME ZONE (default): Specifies that input-string does not include a time zone. If the input string contains a time zone, Vertica ignores this qualifier. Instead, it conforms to WITH TIME ZONE behavior.</td>
</tr>
<tr>
<td></td>
<td>- WITH TIME ZONE: Specifies to convert input-string to UTC, using the UTC offset for the specified time zone. If the input string omits a time zone, Vertica uses the UTC offset of the time zone that is configured for your system.</td>
</tr>
<tr>
<td>input-string</td>
<td>See Input String below.</td>
</tr>
<tr>
<td>AT TIME ZONE zone</td>
<td>See TIME AT TIME ZONE and TIMESTAMP AT TIME ZONE.</td>
</tr>
</tbody>
</table>
TIME versus TIMETZ

TIMETZ and TIMESTAMPTZ are not parallel SQL constructs. TIMESTAMPTZ records a time and date in GMT, converting from the specified TIME ZONE. TIMETZ records the specified time and the specified time zone, in minutes, from GMT.

Limits

<table>
<thead>
<tr>
<th>Name</th>
<th>Low Value</th>
<th>High Value</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME [p]</td>
<td>00:00:00.00</td>
<td>23:59:60.999999</td>
<td>1 µs</td>
</tr>
<tr>
<td>TIME [p] WITH TIME ZONE</td>
<td>00:00:00.00+14</td>
<td>23:59:59.999999-14</td>
<td>1 µs</td>
</tr>
</tbody>
</table>

Input String

A TIME input string can be set to any of the formats shown below:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04:05:06</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05:06</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>040506</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05 AM</td>
<td>Same as 04:05; AM does not affect value</td>
</tr>
<tr>
<td>04:05 PM</td>
<td>Same as 16:05</td>
</tr>
<tr>
<td>04:05:06.789-8</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05:06-08:00</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05-08:00</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>040506-08</td>
<td>ISO 8601</td>
</tr>
<tr>
<td>04:05:06 PST</td>
<td>Time zone specified by name</td>
</tr>
</tbody>
</table>
Data Type Coercion

You can cast a TIME or TIMETZ interval to a TIMESTAMP. This returns the local date and time as follows:

```sql
=> SELECT (TIME '3:01am')::TIMESTAMP;
--column--
2012-08-30 03:01:00
(1 row)

=> SELECT (TIMETZ '3:01am')::TIMESTAMP;
--column--
2012-08-22 03:01:00
(1 row)
```

Casting the same TIME or TIMETZ interval to a TIMESTAMPTZ returns the local date and time, appended with the UTC offset—in this example, -05:

```sql
=> SELECT (TIME '3:01am')::TIMESTAMPTZ;
--column--
2016-12-08 03:01:00-05
(1 row)
```

TIME AT TIME ZONE

Converts the specified TIME to the time in another time zone.

Syntax

```
TIME [WITH TIME ZONE] 'input-string' AT TIME ZONE 'zone'
```

Parameters

<table>
<thead>
<tr>
<th>WITH TIME ZONE</th>
<th>Converts the input string to UTC, using the UTC offset for the specified time zone. If the input string omits a time zone, Vertica uses the UTC offset of the time zone that is configured for your system, and converts the input string accordingly</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone</td>
<td>Specifies the time zone to use in the conversion, either as a literal or</td>
</tr>
</tbody>
</table>
interval that specifies UTC offset:

- AT TIME ZONE INTERVAL 'utc-offset'
- AT TIME ZONE 'time-zone-literal'

For details, see Specifying Time Zones below.

Note: Vertica treats literals TIME ZONE and TIMEZONE as synonyms.

Specifying Time Zones

You can specify time zones in two ways:

- A string literal such as America/Chicago or PST
- An interval that specifies a UTC offset—for example, INTERVAL '-08:00'

It is generally good practice to specify time zones with literals that indicate a geographic location. Vertica makes the necessary seasonal adjustments, and thereby avoids inconsistent results. For example, the following two queries are issued when daylight time is in effect. Because the local UTC offset during daylight time is -04, both queries return the same results:

```sql
=> SELECT CURRENT_TIME(0) "EDT";
    EDT
  ---------
  12:34:35-04
  (1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE 'America/Denver' "Mountain Time";
    Mountain Time
  ---------
  10:34:35-06
  (1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE INTERVAL '-06:00' "Mountain Time";
    Mountain Time
  ---------
  10:34:35-06
  (1 row)
```

If you issue a use the UTC offset in a similar query when standard time is in effect, you must adjust the UTC offset accordingly—for Denver time, to -07—otherwise, Vertica returns a different (and erroneous) result:
=> SELECT CURRENT_TIME(0) "EST";

EST
-------------
14:18:22-05
(1 row)

=> SELECT CURRENT_TIME(0) AT TIME ZONE INTERVAL '-06:00' "Mountain Time";

Mountain Time
-------------
13:18:22-06
(1 row)

You can show and set the session's time zone with `SHOW TIMEZONE` and `SET TIME ZONE`, respectively:

=> SHOW TIMEZONE;

name     setting
-------------
timezone  America/New_York
(1 row)

=> SELECT CURRENT_TIME(0) "Eastern Daylight Time";

Eastern Daylight Time
-------------
12:18:24-04
(1 row)

=> SET TIMEZONE 'America/Los_Angeles';

SET

=> SELECT CURRENT_TIME(0) "Pacific Daylight Time";

Pacific Daylight Time
-------------
09:18:24-07
(1 row)

Time Zone Literals

To view the default list of valid literals, see the files in the following directory:

opt/vertica/share/timezonesets

For example:

$ cat Antarctica.txt
...
# src/timezone/tznames/Antarctica.txt
#
AWST 28800  # Australian Western Standard Time
#   (Antarctica/Casey)
#   (Australia/Perth)
...
Examples

The following example assumes that local time is EST (Eastern Standard Time). The query converts the specified time to MST (mountain standard time):

```sql
=> SELECT CURRENT_TIME();
    timezone
----------
10:10:56-05
(1 row)

=> SELECT TIME '10:10:56' AT TIME ZONE 'America/Denver' "Denver Time";
Denver Time
----------
08:10:56-07
(1 row)
```

The next example adds a time zone literal to the input string—in this case, Europe/Vilnius—and converts the time to MST:

```sql
=> SELECT TIME '09:56:13 Europe/Vilnius' AT TIME ZONE 'America/Denver';
Denver Time
----------
00:56:13-07
(1 row)
```
TIMESTAMP/TIMESTAMPTZ

Stores the specified date and time. TIMESTAMPTZ is the same as TIMESTAMP WITH TIME ZONE: both data types store the UTC offset of the specified time.
TIMESTAMP is an alias for DATETIME and SMALLDATETIME.

Syntax

**TIMESTAMP**

TIMESTAMP [ (p) ] [ { WITHOUT | WITH } TIME ZONE ] 'input-string' [ AT TIME ZONE zone ]

**TIMESTAMPTZ**

TIMESTAMPTZ [ (p) ] 'input-string' [ AT TIME ZONE zone ]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p</strong></td>
<td>Optional precision value that specifies the number of fractional digits retained in the seconds field, an integer value between 0 and 6. If you omit specifying precision, Vertica returns up to 6 fractional digits.</td>
</tr>
<tr>
<td><strong>WITHOUT TIME ZONE</strong></td>
<td>Specifies whether to include a time zone with the stored value:</td>
</tr>
<tr>
<td><strong>WITH TIME ZONE</strong></td>
<td>Specifies whether to include a time zone with the stored value:</td>
</tr>
<tr>
<td><strong>input-string</strong></td>
<td>See Input String below.</td>
</tr>
<tr>
<td><strong>AT TIME ZONE zone</strong></td>
<td>See TIMESTAMP AT TIME ZONE.</td>
</tr>
</tbody>
</table>
Limits

In the following table, values are rounded. See Date/Time Data Types for more detail.

<table>
<thead>
<tr>
<th>Name</th>
<th>Low Value</th>
<th>High Value</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP [ (p) ] [ WITHOUT TIME ZONE ]</td>
<td>290279 BC</td>
<td>294277 AD</td>
<td>1 µs</td>
</tr>
<tr>
<td>TIMESTAMP [ (p) ] WITH TIME ZONE</td>
<td>290279 BC</td>
<td>294277 AD</td>
<td>1 µs</td>
</tr>
</tbody>
</table>

Input String

The date/time input string concatenates a date and a time. The input string can include a time zone, specified as a literal such as America/Chicago, or as a UTC offset.

The following list represents typical date/time input variations:

- 1999-01-08 04:05:06
- 1999-01-08 04:05:06 -8:00
- January 8 04:05:06 1999 PST

The input string can also specify the calendar era, either AD (default) or BC. If you omit the calendar era, Vertica assumes the current calendar era (AD). The calendar era typically follows the time zone; however, the input string can include it in various locations. For example, the following queries return the same results:

```sql
=> SELECT TIMESTAMP WITH TIME ZONE 'March 1, 44 12:00 CET' "Caesar's Time of Death EST";
Caesar's Time of Death EST
---------------------------------------------
0044-03-01 06:00:00-05 BC
(1 row)
```

```sql
=> SELECT TIMESTAMP WITH TIME ZONE 'March 1, 44 12:00 BC CET' "Caesar's Time of Death EST";
Caesar's Time of Death EST
---------------------------------------------
0044-03-01 06:00:00-05 BC
(1 row)
```
Examples: TIMESTAMP Computation

<table>
<thead>
<tr>
<th>Statement</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01');</td>
<td>16 10:10</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') / 7;</td>
<td>2 08:17:08.571429</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2009-05-29 15:21:00.456789' - TIMESTAMP '2009-05-28')(3);</td>
<td>1 15:21:00.457</td>
</tr>
<tr>
<td>SELECT '2017-03-18 07:00'::TIMESTAMPTZ() + INTERVAL '1.5 day';</td>
<td>2017-03-19 19:00:00-04</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') day;</td>
<td>16</td>
</tr>
<tr>
<td>SELECT floor((TIMESTAMP '2014-01-17 10:00' - TIMESTAMP '2014-01-01') / interval '7');</td>
<td>2</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') second;</td>
<td>141660.456789</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') year;</td>
<td>3</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') month;</td>
<td>40</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') year to month;</td>
<td>3-4</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') second (3);</td>
<td>107536860.457</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') minute;</td>
<td>1792281</td>
</tr>
<tr>
<td>SELECT (TIMESTAMP '2012-05-29 15:21:00.456789' - TIMESTAMP '2009-01-01') minute to second(3);</td>
<td>1792281:00.457</td>
</tr>
<tr>
<td>SELECT TIMESTAMP 'infinity';</td>
<td>infinity</td>
</tr>
</tbody>
</table>

**TIMESTAMP AT TIME ZONE**

Converts the specified TIMESTAMP or TIMESTAMPTZ (TIMESTAMP WITH TIMEZONE) to another time zone. Vertica executes AT TIME ZONE differently, depending on whether the date input is a TIMESTAMP or TIMESTAMPTZ. See [TIMESTAMP Versus TIMESTAMPTZ Behavior](#) below.
Syntax

\[ \text{timestamp-clause} \ AT \ TIME \ ZONE \ 'zone' \]

Parameters

<table>
<thead>
<tr>
<th>\textit{timestamp-clause}</th>
<th>Specifies the timestamp to convert, either \texttt{TIMESTAMP} or \texttt{TIMESTAMPTZ}. For details, see \texttt{TIMESTAMP/TIMESTAMPTZ}.</th>
</tr>
</thead>
</table>
| AT TIME ZONE \textit{zone} | Specifies the time zone to use in the timestamp conversion, where \textit{zone} is a literal or interval that specifies a UTC offset:  
  - AT TIME ZONE INTERVAL \texttt{utc-offset}'  
  - AT TIME ZONE \texttt{time-zone-literal}'  
  For details, see \texttt{Specifying Time Zones} below. |

Note: Vertica treats literals \texttt{TIME ZONE} and \texttt{TIMEZONE} as synonyms.

TIMESTAMP Versus TIMESTAMPTZ Behavior

How Vertica interprets \texttt{AT TIME ZONE} depends on whether the date input is a \texttt{TIMESTAMP} or \texttt{TIMESTAMPTZ}:

<table>
<thead>
<tr>
<th>Date input</th>
<th>Action</th>
</tr>
</thead>
</table>
| TIMESTAMP    | If the input string specifies no time zone, Vertica performs two actions:  
  1. Converts the input string to the time zone of the AT TIME ZONE argument.  
  2. Returns the time for the current session's time zone. If the input string includes a time zone, Vertica implicitly casts it to a TIMESTAMPTZ and converts it accordingly (see TIMESTAMPTZ below).  
  For example, the following statement specifies a TIMESTAMP with no time zone input:  
  ```sql
  SELECT AT TIME ZONE 'America/New_York' '2023-01-01'
  ```  
  This statement will return the time in the America/New_York time zone on January 1, 2023. |
| TIMESTAMPTZ  | If the input string cannot be parsed as a time zone, Verica implicitly casts it to a TIMESTAMPTZ and converts it accordingly. For example, the following statement specifies a TIMESTAMPTZ with no time zone input:  
  ```sql
  SELECT AT TIME ZONE '2023-01-01'
  ```  
  This statement will return the time for the current session's time zone on January 1, 2023. |
<table>
<thead>
<tr>
<th>Date input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone. Vertica executes the statement as follows:</td>
<td></td>
</tr>
<tr>
<td>1. Converts the input string to PDT (Pacific Daylight Time).</td>
<td></td>
</tr>
<tr>
<td>2. Returns that time in the local time zone, which is three hours later:</td>
<td></td>
</tr>
</tbody>
</table>

```sql
=> SHOW TIMEZONE;
   name | setting
---------------------
timezone | America/New_York
(1 row)

SELECT TIMESTAMP '2017-3-14 5:30' AT TIME ZONE 'PDT';
   timezone
---------------------
2017-03-14 08:30:00-04
(1 row)
```

**TIMESTAMPZ**

Vertica converts the input string to the time zone of the AT TIME ZONE argument and returns that time.

For example, the following statement specifies a TIMESTAMPTZ data type. The input string omits any time zone expression, so Vertica assumes the input string to be in local time zone (America/New_York) and returns the time of the AT TIME ZONE argument:

```sql
=> SHOW TIMEZONE;
   name | setting
---------------------
timezone | America/New_York
(1 row)

=> SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40' AT TIME ZONE 'America/Denver';
   timezone
---------------------
2001-02-16 18:38:40
(1 row)
```

The input string in the next statement explicitly specifies a time zone, so Vertica coerces the TIMESTAMP to a TIMESTAMPTZ and returns the time of the AT TIME ZONE argument:

```sql
=> SELECT TIMESTAMP '2001-02-16 20:38:40 America/Mexico_City' AT TIME ZONE 'Asia/Tokyo';
   timezone
---------------------
2001-02-17 11:38:40
(1 row)
```
Specifying Time Zones

You can specify time zones in two ways:

- A string literal such as America/Chicago or PST
- An interval that specifies a UTC offset—for example, INTERVAL ' -08:00'

It is generally good practice to specify time zones with literals that indicate a geographic location. Vertica makes the necessary seasonal adjustments, and thereby avoids inconsistent results. For example, the following two queries are issued when daylight time is in effect. Because the local UTC offset during daylight time is -04, both queries return the same results:

```
=> SELECT TIMESTAMPTZ '2017-03-16 09:56:13' AT TIME ZONE 'America/Denver' "Denver Time";

<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>America/Denver</td>
</tr>
</tbody>
</table>

Denver Time
2017-03-16 07:56:13
(1 row)
```

```
=> SELECT TIMESTAMPTZ '2017-03-16 09:56:13' AT TIME ZONE INTERVAL '-06:00' "Denver Time";

<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>America/Denver</td>
</tr>
</tbody>
</table>

Denver Time
2017-03-16 07:56:13
(1 row)
```

If you issue a use the UTC offset in a similar query when standard time is in effect, you must adjust the UTC offset accordingly—for Denver time, to -07—otherwise, Vertica returns a different (and erroneous) result:

```
=> SELECT TIMESTAMPTZ '2017-01-16 09:56:13' AT TIME ZONE 'America/Denver' "Denver Time";

<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>America/Denver</td>
</tr>
</tbody>
</table>

Denver Time
2017-01-16 07:56:13
(1 row)
```

```
=> SELECT TIMESTAMPTZ '2017-01-16 09:56:13' AT TIME ZONE INTERVAL '-06:00' "Denver Time";

<table>
<thead>
<tr>
<th>Timezone</th>
</tr>
</thead>
<tbody>
<tr>
<td>America/Denver</td>
</tr>
</tbody>
</table>

Denver Time
2017-01-16 08:56:13
(1 row)
```

You can show and set the session's time zone with `SHOW TIMEZONE` and `SET TIME ZONE`, respectively:

```
=> SHOW TIMEZONE;

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>timezone</td>
<td>America/New_York</td>
</tr>
</tbody>
</table>

(1 row)
```
=> SELECT CURRENT_TIMESTAMP(0) "Eastern Daylight Time";
Eastern Daylight Time
------------------------
2017-03-20 12:18:24-04
(1 row)
=> SET TIMEZONE 'America/Los_Angeles';
SET
=> SELECT CURRENT_TIMESTAMP(0) "Pacific Daylight Time";
Pacific Daylight Time
------------------------
2017-03-20 09:18:24-07
(1 row)

Time Zone Literals

To view the default list of valid literals, see the files in the following directory:

opt/vertica/share/timezonesets

For example:

$ cat Antarctica.txt
...
# src/timezone/tznames/Antarctica.txt
#
AWST 28800 # Australian Western Standard Time
  # (Antarctica/Casey)
  # (Australia/Perth)
...
NZST 43200 # New Zealand Standard Time
  # (Antarctica/McMurdo)
  # (Pacific/Auckland)
ROTT -10800 # Rothera Time
  # (Antarctica/Rothera)
SYOT 10800 # Syowa Time
  # (Antarctica/Syowa)
VOST 21600 # Vostok time
  # (Antarctica/Vostok)

Examples

The following examples assume that the current time zone is America/New_York:
Convert the string 10:00 to the specified time zone:

=> SELECT TIME '10:00' AT TIME ZONE 'America/Chicago';
timezone
           10:00
Long Data Types

Store data up to 32000000 octets. Vertica supports two long data types:

- **LONG VARBINARY**: Variable-length raw-byte data, such as spatial data. LONG VARBINARY values are not extended to the full width of the column.
- **LONG VARCHAR**: Variable-length strings, such as log files and unstructured data. LONG VARCHAR values are not extended to the full width of the column.

Use LONG data types only when you need to store data greater than the maximum size of VARBINARY and VARCHAR data types (65 KB). Long data can include unstructured data, online comments or posts, or small log files.

Flex tables have a default LONG VARBINARY __raw__ column, with a NOT NULL constraint. For more information, see Using Flex Tables.

Syntax

**LONG VARBINARY**

LONG VARBINARY (max-Length)

**LONG VARCHAR**

LONG VARCHAR (octet-Length)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>max-Length</strong></td>
<td>Specifies the length of the byte string or column width, declared in bytes (octets). Maximum value: 32000000 Default value: 1 MB</td>
</tr>
<tr>
<td><strong>octet-Length</strong></td>
<td>Specifies the length of the string or column width, declared in bytes (octets).</td>
</tr>
</tbody>
</table>
Optimized Performance

For optimal performance of LONG data types, Vertica recommends that you:

- Use the LONG data types as *storage only* containers; Vertica supports operations on the content of LONG data types, but does not support all the operations that VARCHAR and VARBINARY take.

- Use VARBINARY and VARCHAR data types, instead of their LONG counterparts, whenever possible. VARBINARY and VARCHAR data types are more flexible and have a wider range of operations.

- Do not sort, segment, or partition projections on LONG data type columns.

- Do not add constraints, such as a primary key, to any LONG VARBINARY or LONG VARCHAR columns.

- Do not join or aggregate any LONG data type columns.

Example

The following example creates a table `user_comments` with a LONG VARCHAR column and inserts data into it:

```sql
=> CREATE TABLE user_comments
(id INTEGER,
 username VARCHAR(200),
 time_posted TIMESTAMP,
 comment_text LONG VARCHAR(200000));

=> INSERT INTO user_comments VALUES
(1, 'User1',
 'The weather tomorrow will be cold and rainy and then on the day after, the sun will come and the temperature will rise dramatically.');
```
Numeric Data Types

Numeric data types are numbers stored in database columns. These data types are typically grouped by:

- **Exact** numeric types, values where the precision and scale need to be preserved. The exact numeric types are INTEGER, BIGINT, DECIMAL, NUMERIC, NUMBER, and MONEY.

- **Approximate** numeric types, values where the precision needs to be preserved and the scale can be floating. The approximate numeric types are DOUBLE PRECISION, FLOAT, and REAL.

Implicit casts from INTEGER, FLOAT, and NUMERIC to VARCHAR are not supported. If you need that functionality, write an explicit cast using one of the following forms:

```
CAST(numeric-expression AS data-type)
numeric-expression::data-type
```

For example, you can cast a float to an integer as follows:

```
=> SELECT (FLOAT '123.5')::INT;
  column?
-------------------
    124
(1 row)
```

String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- **Scientific notation**:

  ```
  => SELECT FLOAT '1e10';
  ?column?
  -------------------
   10000000000
(1 row)
  ```

- **BINARY scaling**:

  ```
  => SELECT NUMERIC '1p10';
  ?column?
  ------------
   1024
(1 row)
  ```

- **hexadecimal**:
=> SELECT NUMERIC '0x0abc';
?column?
--------
  2748
(1 row)

DOUBLE PRECISION (FLOAT)

Vertica supports the numeric data type DOUBLE PRECISION, which is the IEEE-754 8-byte floating point type, along with most of the usual floating point operations.

Syntax

[ DOUBLE PRECISION | FLOAT | FLOAT(n) | FLOAT8 | REAL ]

Parameters

Note: On a machine whose floating-point arithmetic does not follow IEEE-754, these values probably do not work as expected.

Double precision is an inexact, variable-precision numeric type. In other words, some values cannot be represented exactly and are stored as approximations. Thus, input and output operations involving double precision might show slight discrepancies.

- All of the DOUBLE PRECISION data types are synonyms for 64-bit IEEE FLOAT.
- The \( n \) in FLOAT\((n)\) must be between 1 and 53, inclusive, but a 53-bit fraction is always used. See the IEEE-754 standard for details.
- For exact numeric storage and calculations (money for example), use NUMERIC.
- Floating point calculations depend on the behavior of the underlying processor, operating system, and compiler.
- Comparing two floating-point values for equality might not work as expected.
- While Vertica treats decimal values as FLOAT internally, if a column is defined as FLOAT then you cannot read decimal values from ORC and Parquet files. In those formats, FLOAT and DECIMAL are different types.
Values

COPY accepts floating-point data in the following format:

- Optional leading white space
- An optional plus ("+") or minus sign ("-"")
- A decimal number, a hexadecimal number, an infinity, a NAN, or a null value

Decimal Number

A decimal number consists of a non-empty sequence of decimal digits possibly containing a radix character (decimal point "."), optionally followed by a decimal exponent. A decimal exponent consists of an "E" or "e", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 10.

Hexadecimal Number

A hexadecimal number consists of a "0x" or "0X" followed by a non-empty sequence of hexadecimal digits possibly containing a radix character, optionally followed by a binary exponent. A binary exponent consists of a "P" or "p", followed by an optional plus or minus sign, followed by a non-empty sequence of decimal digits, and indicates multiplication by a power of 2. At least one of radix character and binary exponent must be present.

Infinity

An infinity is either INF or INFINITY, disregarding case.

NaN (Not A Number)

A NaN is NAN (disregarding case) optionally followed by a sequence of characters enclosed in parentheses. The character string specifies the value of NaN in an implementation-dependent manner. (The Vertica internal representation of NaN is 0xfff8000000000000LL on x86 machines.)

When writing infinity or NAN values as constants in a SQL statement, enclose them in single quotes. For example:

```sql
=> UPDATE table SET x = 'Infinity'
```

Note: Vertica follows the IEEE definition of NaNs (IEEE 754). The SQL standards do not specify how floating point works in detail.
IEEE defines NaNs as a set of floating point values where each one is not equal to anything, even to itself. A NaN is not greater than and at the same time not less than anything, even itself. In other words, comparisons always return false whenever a NaN is involved.

However, for the purpose of sorting data, NaN values must be placed somewhere in the result. The value generated 'NaN' appears in the context of a floating point number matches the NaN value generated by the hardware. For example, Intel hardware generates (0xffff800000000000LL), which is technically a Negative, Quiet, Non-signaling NaN. Vertica uses a different NaN value to represent floating point NULL (0x7fffffffffffffffLL). This is a Positive, Quiet, Non-signaling NaN and is reserved by Vertica.

A NaN example follows.

```sql
=> SELECT CBRT('Nan'); -- cube root
   CBRT
    -------
     Nan
     (1 row)
=> SELECT 'Nan' > 1.0;
   ?column?
    1
     (1 row)
```

**Null Value**

The load file format of a null value is user defined, as described in the COPY command. The Vertica internal representation of a null value is 0x7fffffffffffffffLL. The interactive format is controlled by the vsql printing option null. For example:

```
\pset null '(null)'
```

The default option is not to print anything.

**Rules**

- \-0 == +0
- 1/0 = Infinity
- 0/0 == Nan
- NaN != anything (even NaN)

To search for NaN column values, use the following predicate:
... WHERE column != column

This is necessary because WHERE column = 'Nan' cannot be true by definition.

Sort Order (Ascending)

- NaN
- -Inf
- numbers
- +Inf
- NULL

Notes

- NULL appears last (largest) in ascending order.
- All overflows in floats generate +/-infinity or NaN, per the IEEE floating point standard.

INTEGER

A signed 8-byte (64-bit) data type.

Syntax

[ INTEGER | INT | BIGINT | INT8 | SMALLINT | TINYINT ]

Parameters

INT, INTEGER, INT8, SMALLINT, TINYINT, and BIGINT are all synonyms for the same signed 64-bit integer data type. Automatic compression techniques are used to conserve disk space in cases where the full 64 bits are not required.
Notes

- The range of values is $-2^{63}+1$ to $2^{63}-1$.
- $2^{63} = 9,223,372,036,854,775,808$ (19 digits).
- The value $-2^{63}$ is reserved to represent NULL.
- NULL appears first (smallest) in ascending order.
- Vertica does not have an explicit 4-byte (32-bit integer) or smaller types. Vertica's encoding and compression automatically eliminate the storage overhead of values that fit in less than 64 bits.

Restrictions

- The JDBC type INTEGER is 4 bytes and is not supported by Vertica. Use BIGINT instead.
- Vertica does not support the SQL/JDBC types NUMERIC, SMALLINT, or TINYINT.
- Vertica does not check for overflow (positive or negative) except in the aggregate function SUM(). If you encounter overflow when using SUM, use SUM_FLOAT(), which converts to floating point.

See Also

Data Type Coercion Chart

NUMERIC

Numeric data types store fixed-point numeric data. For example, a value of $123.45$ can be stored in a NUMERIC(5,2) field.

Syntax

```
numeric-type [ ( precision[, scale] ) ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric-type</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- NUMERIC</td>
</tr>
<tr>
<td></td>
<td>- DECIMAL</td>
</tr>
<tr>
<td></td>
<td>- NUMBER</td>
</tr>
<tr>
<td></td>
<td>- MONEY</td>
</tr>
<tr>
<td>precision</td>
<td>An unsigned integer that specifies the total number of significant digits that the data type stores, where <code>precision</code> is ≤ 1024. If omitted, the default precision depends on numeric type that you specify. If you assign a value that exceeds <code>precision</code>, Vertica returns an error. If a data type's precision is ≤ 18, performance is equivalent to an INTEGER data type, regardless of scale. When possible, Vertica recommends using a precision ≤ 18.</td>
</tr>
<tr>
<td>scale</td>
<td>An unsigned integer that specifies the maximum number of digits to the right of the decimal point to store. <code>scale</code> must be ≤ <code>precision</code>. If omitted, the default scale depends on numeric type that you specify. If you assign a value with more decimal digits than <code>scale</code>, the scale is rounded to <code>scale</code> digits.</td>
</tr>
</tbody>
</table>

Maximum Significant Digits

The maximum number of significant digits (digits to the left of the decimal point) is equal to precision minus scale. For example, NUMERIC(9, 6) allows three significant digits. In this case, if you insert a value that has more than three significant digits, an error occurs.

Default Precision and Scale

NUMERIC, DECIMAL, NUMBER, and MONEY differ in their default precision and scale values:
### Numeric Versus Integer and Floating Data Types

Numeric data types are *exact* data types that store values of a specified precision and scale, expressed with a number of digits before and after a decimal point. This contrasts with the Vertica integer and floating data types:

- **DOUBLE PRECISION (FLOAT)** supports ~15 digits, variable exponent, and represents numeric values approximately. It can be less precise than NUMERIC data types.

- **INTEGER** supports ~18 digits, whole numbers only.

The NUMERIC data type is preferred for non-integer constants, because it is always exact. For example:

```
=> SELECT 1.1 + 2.2 = 3.3;
?column?
---------
t
(1 row)

=> SELECT 1.1::float + 2.2::float = 3.3::float;
?column?
---------
f
(1 row)
```

### Numeric Operations

Supported numeric operations include the following:

<table>
<thead>
<tr>
<th>Basic math</th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>SUM</td>
<td>MIN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
NUMERIC divide operates directly on numeric values, without converting to floating point. The result has at least 18 decimal places and is rounded.

NUMERIC mod (including %) operates directly on numeric values, without converting to floating point. The result has the same scale as the numerator and never needs rounding.

Some complex operations used with numeric data types result in an implicit cast to FLOAT. When using SQRT, STDDEV, transcendental functions such as LOG, and TO_CHAR/TO_NUMBER formatting, the result is always FLOAT.

Examples

The following series of commands creates a table that contains a numeric data type and then performs some mathematical operations on the data:

```sql
=> CREATE TABLE num1 (id INTEGER, amount NUMERIC(8,2));
```

Insert some values into the table:

```sql
=> INSERT INTO num1 VALUES (1, 123456.78);
```

Query the table:

```sql
=> SELECT * FROM num1;
id | amount
---|--------
1  | 123456.78
(1 row)
```

The following example returns the NUMERIC column, amount, from table num1:

```sql
=> SELECT amount FROM num1;
amount
--------
```

The following syntax adds one (1) to the amount:

```sql
=> SELECT amount+1 AS 'amount' FROM num1;

<table>
<thead>
<tr>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>123457.78</td>
</tr>
</tbody>
</table>
```

The following syntax multiplies the amount column by 2:

```sql
=> SELECT amount*2 AS 'amount' FROM num1;

<table>
<thead>
<tr>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>246913.56</td>
</tr>
</tbody>
</table>
```

The following syntax returns a negative number for the amount column:

```sql
=> SELECT -amount FROM num1;

<table>
<thead>
<tr>
<th>?column?</th>
</tr>
</thead>
<tbody>
<tr>
<td>-123456.78</td>
</tr>
</tbody>
</table>
```

The following syntax returns the absolute value of the amount argument:

```sql
=> SELECT ABS(amount) FROM num1;

<table>
<thead>
<tr>
<th>ABS</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456.78</td>
</tr>
</tbody>
</table>
```

The following syntax casts the NUMERIC amount as a FLOAT data type:

```sql
=> SELECT amount::float FROM num1;

<table>
<thead>
<tr>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456.78</td>
</tr>
</tbody>
</table>
```

See Also

- Mathematical Functions
- Valid Encoding for Numeric Data Types
Numeric Data Type Overflow

Vertica does not check for overflow (positive or negative) except in the aggregate function SUM ( ). If you encounter overflow when using SUM, use SUM_FLOAT ( ) which converts to floating point.

For a detailed discussion of how Vertica handles overflow when you use the functions SUM, SUM_FLOAT, and AVG with numeric data types, see Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG. The discussion includes directives for turning off silent numeric overflow and setting precision for numeric data types.

Dividing by zero returns an error:

```sql
=> SELECT 0/0;
ERROR 3117: Division by zero

=> SELECT 0.0/0;
ERROR 3117: Division by zero

=> SELECT 0 // 0;
ERROR 3117: Division by zero

=> SELECT 200.0/0;
ERROR 3117: Division by zero

=> SELECT 116.43 // 0;
ERROR 3117: Division by zero
```

Dividing zero as a FLOAT by zero returns NaN:

```sql
=> SELECT 0.0::float/0;
?column?
---------
NaN

=> SELECT 0.0::float//0;
?column?
---------
NaN
```

Dividing a non-zero FLOAT by zero returns Infinity:

```sql
=> SELECT 2.0::float/0;
?column?
---------
Infinity

=> SELECT 200.0::float//0;
?column?
---------
Infinity
```
Add, subtract, and multiply operations ignore overflow. Sum and average operations use 128-bit arithmetic internally. SUM() reports an error if the final result overflows, suggesting the use of SUM_FLOAT(INT), which converts the 128-bit sum to a FLOAT. For example:

```sql
=> CREATE TEMP TABLE t (i INT);
=> INSERT INTO t VALUES (1<<62);
=> INSERT INTO t VALUES (1<<62);
=> INSERT INTO t VALUES (1<<62);
=> SELECT SUM(i) FROM t;
  ERROR: sum() overflowed
  HINT: try sum_float() instead
=> SELECT SUM_FLOAT(i) FROM t;
  SUM_FLOAT
  2.30584300921369e+19
```

**Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG**

When you use the functions SUM, SUM_FLOAT, and AVG with a NUMERIC data type, be aware that overflow can occur and how Vertica responds to that overflow.

This discussion applies to both the aggregate and analytic functions.

For queries, when using the functions SUM, SUM_FLOAT, and AVG with a NUMERIC data type, Vertica allows for silent overflow if you exceed your specified precision.

Vertica also allows numeric overflow when you use the SUM or SUM_FLOAT functions with LAPs.

**Default Overflow Handling**

With NUMERIC data types, Vertica internally works with multiples of 18 digits. If your specified precision is less than 18 (for example, x(12, 0)), Vertica allows for an overflow up to and including the first multiple of 18. In some situations, if you sum a column (SUM(x)), you can exceed the number of digits Vertica internally reserves for the result. In this case, Vertica allows a silent overflow.

**Turning Off Silent Numeric Overflow**

You can turn off silent numeric overflow and instruct Vertica to implicitly include extra digit places. Specifying extra spaces allows Vertica to consistently return your expected results, even
when you exceed the precision specified in your DDL.

You turn off silent numeric overflow by setting the parameter AllowNumericOverflow to 0 (false).

When you set the parameter to 0, Vertica considers the value of a corresponding parameter, NumericSumExtraPrecisionDigits.

The NumericSumExtraPrecisionDigits parameter defaults to 6, meaning that Vertica internally add six places beyond your DDL-specified precision. Adding extra precision digits can allow Vertica to consistently return results that overflow your DDL-specified precision. However, there can be a performance impact for crossing into the second multiple of 18 internally.

An example:

- Suppose your DDL specifies 11 (for example, `x(11, 0)`) and you accept the default of NumericSumExtraPrecisionDigits (6). In this case, Vertica internally stays within the first multiple of 18 digits and no additional performance impact occurs.

- Given the same example, if you set NumericSumExtraPrecisionDigits to 10, Vertica internally crosses a threshold into the second multiple of 18. Performance-wise, if (hypothetically) the first example is performance “a,” then the second is “2a,” substantially increasing the performance impact. Beyond the second multiple of 18, the performance impact continues to be "2a."

This sample representation shows how Vertica responds internally when you set AllowNumericOverflow to 0 (false).
Vertica recommends that you turn off silent numeric overflow and set the parameter NumericSumExtraPrecisionDigits if you expect to exceed the precision specified in your DDL. Crossing into the second multiple of 18 can affect performance. Therefore, consider carefully before setting NumericSumExtraPrecisionDigits to a number higher than what you need for returning the SUM of your numeric columns.

Be aware that, if you turn off AllowNumericOverflow, and you exceed the number of extra precision digits set by NumericSumExtraPrecisionDigits, Vertica returns an error.

Impact on Live Aggregate Projections (LAPs)

For LAPs, Vertica also allows silent numeric overflow if your LAP uses the SUM or SUM_FLOAT functions. To turn off silent numeric overflow for LAPs:

1. Set the parameter AllowNumericOverflow to 0.
2. Set the parameter NumericSumExtraPrecisionDigits to the number of implicit digits you want. Alternatively, use the default setting of 6.
3. Drop and re-create your LAPs.

If you turn off silent numeric overflow, be aware of the following scenarios where an overflow causes a roll back or error message. In these examples, AllowNumericOverflow is set to 0 (false), and each LAP uses the SUM or SUM_FLOAT function.

When numeric overflow is off:

- A load can roll back upon overflow.
  
  Vertica aggregates data before loading in into a LAP. If you are inserting, copying, or merging data, and an overflow occurs during load as Vertica is aggregating the data, Vertica rolls back the load.

- An overflow can occur after load as Vertica sums existing data.
  
  Vertica computes the sum of existing data separately from the computation that it does during data load. If your LAP selects a column using SUM or SUM_FLOAT and an overflow occurs, Vertica produces an error message. This response is similar to the way Vertica produces an error for a query using the SUM or SUM_FLOAT function.

- An overflow can occur during merge-out.
Vertica logs a message during merge-out if an overflow occurs as Vertica computes a final sum during the tuple mover operation. If an error occurs, Vertica marks the LAP as out-of-date. Vertica no longer runs tuple mover operations with the out-of-date LAP.

**Spatial Data Types**

Vertica supports two spatial data types. These data types store two- and three-dimensional spatial objects in a table column:

- **GEOMETRY**: Spatial object with coordinates expressed as \((x,y)\) pairs, defined in the Cartesian plane. All calculations use Cartesian coordinates.

- **GEOGRAPHY**: Spatial object defined as on the surface of a perfect sphere, or a spatial object in the WGS84 coordinate system. Coordinates are expressed in longitude/latitude angular values, measured in degrees. All calculations are in meters. For perfect sphere calculations, the sphere has a radius of 6371 kilometers, which approximates the shape of the earth.

  **Note**: Some spatial programs use an ellipsoid to model the earth, resulting in slightly different data.

The maximum size of a GEOMETRY or GEOGRAPHY data type is 10,000,000 bytes (10 MB). You cannot use either data type as a table's primary key.

**Syntax**

- **GEOMETRY**
  
  ```sql
  GEOMETRY
  GEOMETRY [ (Length) ]
  ```

- **GEOGRAPHY**
  
  ```sql
  GEOGRAPHY
  GEOGRAPHY [ (Length) ]
  ```

**Parameters**

| Length | The maximum amount of spatial data that a GEOMETRY or GEOGRAPHY column can store |
**UUID Data Type**

Stores universally unique identifiers (UUIDs). UUIDs are 16-byte (128-bit) numbers used to uniquely identify records. To generate UUIDs, Vertica provides the function `UUID_GENERATE`, which returns UUIDs based on high-quality randomness from `/dev/urandom`.

**Syntax**

```
UUID
```

**UUID Input and Output Formats**

UUIDs support input of case-insensitive string literal formats, as specified by RFC 4122. In general, a UUID is written as a sequence of hexadecimal digits, in several groups optionally separated by hyphens, for a total of 32 digits representing 128 bits.

The following input formats are valid:

```
6bbf0744-74b4-46b9-bb05-53905d4538e7
(6bbf0744-74b4-46b9-bb05-53905d4538e7)
6Bf074474044689880553905D4538E7
6BF-0744-74b4-46b9-bb05-5390-5D45-38E7
```

On output, Vertica always uses the following format:

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

For example, the following table defines column `cust_id` as a UUID:

```
=> CREATE TABLE public.Customers
  (  
      cust_id uuid,
      lname varchar(36),
      fname varchar(24)
  );
```

The following input for `cust_id` uses several valid formats:

```
6bbf0744-74b4-46b9-bb05-53905d4538e7
(6bbf0744-74b4-46b9-bb05-53905d4538e7)
6Bf074474044689880553905D4538E7
6BF-0744-74b4-46b9-bb05-5390-5D45-38E7
```
On querying this table, Vertica formats all cust_id data in the same way:

```
=> SELECT cust_id, fname, lname FROM Customers;

<table>
<thead>
<tr>
<th>cust_id</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>9fb01de01-63d-4d09-9415-90e0b4e93b9a</td>
<td>Jeremy</td>
<td>Steinberg</td>
</tr>
<tr>
<td>34462732-ed56-4983-8f3b-e735b0c32d50</td>
<td>Duc</td>
<td>Pham</td>
</tr>
<tr>
<td>cede66b7-3d29-4da6-b700-871fc0ac57be</td>
<td>Thomas</td>
<td>Kearney</td>
</tr>
</tbody>
</table>
```

(3 rows)

Generating UUIDs

You can use the Vertica function `UUID_GENERATE` to automatically generate UUIDs that uniquely identify table records. For example:

```
=> INSERT INTO Customers SELECT UUID_GENERATE(),'Rostova','Natasha';
OUTPUT
-------
1
(1 row)

=> COMMIT;
COMMIT

=> SELECT cust_id, fname, lname FROM Customers;

<table>
<thead>
<tr>
<th>cust_id</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>9fb01de01-63d-4d09-9415-90e0b4e93b9a</td>
<td>Jeremy</td>
<td>Steinberg</td>
</tr>
<tr>
<td>34462732-ed56-4983-8f3b-e735b0c32d50</td>
<td>Duc</td>
<td>Pham</td>
</tr>
<tr>
<td>cede66b7-3d29-4da6-b700-871fc0ac57be</td>
<td>Thomas</td>
<td>Kearney</td>
</tr>
<tr>
<td>9aad6757-fe1b-473a-a10b-b89b7b358c69</td>
<td>Natasha</td>
<td>Rostova</td>
</tr>
</tbody>
</table>
```

(4 rows)

NULL Input and Output

The following string is reserved as NULL for UUID columns:

```
00000000-0000-0000-0000-000000000000
```

Vertica always renders NULL as blank.

The following COPY statements insert NULL values into the UUID column, explicitly and implicitly:
COPY Customers FROM STDIN NULL 'null';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

null | Jane
null | Doe
null | Man
null | Nowhere
null | Doe
null | John
null | Doe

In all cases, Vertica renders NULL as blank:

SELECT cust_id, fname, lname FROM Customers WHERE cust_id IS NULL;

cust_id | fname | lname
---------+-------+-------
          | Nowhere| Man
          | Jane   | Doe
          | John   | Doe
(3 rows)

Usage Restrictions

UUID data types only support relational operators and functions that are also supported by
CHAR and VARCHAR data types—for example, MIN, MAX, and COUNT. UUID data types do not
support mathematical operators or functions, such as SUM and AVG.

Data Type Coercion

Vertica supports two types of data type casting:

- **Implicit casting**: The expression automatically converts the data from one type to another.
- **Explicit casting**: An SQL statement specifies the target data type for the conversion.

Implicit Casting

The ANSI SQL-92 standard supports implicit casting among similar data types:

- Number types
- CHAR, VARCHAR, LONG VARCHAR
- BINARY, VARBINARY, LONG VARBINARY
Vertica supports two types of nonstandard implicit casts:

- **From CHAR to FLOAT**, to match the one from VARCHAR to FLOAT. The following example converts the CHAR '3' to a FLOAT so it can add the number 4.33 to the FLOAT result of the second expression:

  ```sql
  => SELECT '3'::CHAR + 4.33::FLOAT;
  ?column?
  ----------
  7.33
  (1 row)
  ```

- **Between DATE and TIMESTAMP.** The following example DATE to a TIMESTAMP and calculates the time 6 hours, 6 minutes, and 6 seconds back from 12:00 AM:

  ```sql
  => SELECT DATE('now') - INTERVAL '6:6:6';
  ?column?
  ----------------
  2013-07-30 17:53:54
  (1 row)
  ```

When there is no ambiguity about the data type of an expression value, it is implicitly coerced to match the expected data type. In the following command, the quoted string constant '2' is implicitly coerced into an INTEGER value so that it can be the operand of an arithmetic operator (addition):

```sql
=> SELECT 2 + '2';
?column?
----------
 4
(1 row)
```

A concatenate operation explicitly takes arguments of any data type. In the following example, the concatenate operation implicitly coerces the arithmetic expression 2 + 2 and the INTEGER constant 2 to VARCHAR values so that they can be concatenated.

```sql
=> SELECT 2 + 2 || 2;
?column?
----------
42
(1 row)
```

Another example is to first get today's date:

```sql
=> SELECT DATE 'now';
?column?
----------
2013-07-31
(1 row)
```
The following command converts DATE to a TIMESTAMP and adds a day and a half to the results by using INTERVAL:

```sql
=> SELECT DATE 'now' + INTERVAL '1 12:00:00';

---
2013-07-31 12:00:00
(1 row)
```

Most implicit casts stay within their relational family and go in one direction, from less detailed to more detailed. For example:

- DATE to TIMESTAMP/TZ
- INTEGER to NUMERIC to FLOAT
- CHAR to FLOAT
- CHAR to VARCHAR
- CHAR and/or VARCHAR to FLOAT
- CHAR to LONG VARCHAR
- VARCHAR to LONG VARCHAR
- BINARY to VARBINARY
- BINARY to LONG VARBINARY
- VARBINARY to LONG VARBINARY

More specifically, data type coercion works in this manner in Vertica:

<table>
<thead>
<tr>
<th>Type</th>
<th>Direction</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8</td>
<td>&gt;</td>
<td>FLOAT8</td>
<td>Implicit, can lose significance</td>
</tr>
<tr>
<td>FLOAT8</td>
<td>&gt;</td>
<td>INT8</td>
<td>Explicit, rounds</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>&lt;--</td>
<td>CHAR</td>
<td>Implicit, adjusts trailing spaces</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>&lt;--</td>
<td>BINARY</td>
<td>Implicit, adjusts trailing NULs</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>&gt;</td>
<td>LONG VARCHAR</td>
<td>Implicit, adjusts trailing spaces</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>&gt;</td>
<td>LONG VARBINARY</td>
<td>Implicit, adjusts trailing NULs</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>&gt;</td>
<td>LONG VARBINARY</td>
<td>Implicit, adjusts trailing NULs</td>
</tr>
</tbody>
</table>
No other types cast to or from LONGVARBINARY, VARBINARY, or BINARY. In the following list, <any> means one of these types: INT8, FLOAT8, DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, INTERVAL.

- <any> -> VARCHAR—implicit
- VARCHAR -> <any>—explicit, except that VARCHAR->FLOAT is implicit
- <any> <-> CHAR—explicit
- DATE -> TIMESTAMP/TZ—implicit
- TIMESTAMP/TZ -> DATE—explicit, loses time-of-day
- TIME -> TIMETZ—implicit, adds local timezone
- TIMETZ -> TIME—explicit, loses timezone
- TIME -> INTERVAL—implicit, day to second with days=0
- INTERVAL -> TIME—explicit, truncates non-time parts
- TIMESTAMP <-> TIMESTAMPTZ—implicit, adjusts to local timezone
- TIMESTAMP/TZ -> TIME—explicit, truncates non-time parts
- TIMESTAMPTZ -> TIMETZ—explicit
- VARBINARY -> LONG VARBINARY—implicit
- LONG VARBINARY -> VARBINARY—explicit
- VARCHAR -> LONG VARCHAR—implicit
- LONG VARCHAR -> VARCHAR—explicit

**Important:** Implicit casts from INTEGER, FLOAT, and NUMERIC to VARCHAR are not supported. If you need that functionality, write an explicit cast:

```
CAST(x AS data-type-name)
```

or

```
x::data-type-name
```

The following example casts a FLOAT to an INTEGER:

```
=> SELECT(FLOAT '123.5')::INT;
?column?
```
String-to-numeric data type conversions accept formats of quoted constants for scientific notation, binary scaling, hexadecimal, and combinations of numeric-type literals:

- **Scientific notation**:

  ```sql
  => SELECT FLOAT '1e10';
  ?column?
  --------------
   10000000000
  (1 row)
  ```

- **BINARY scaling**:

  ```sql
  => SELECT NUMERIC '1p10';
  ?column?
  1024
  (1 row)
  ```

- **hexadecimal**:

  ```sql
  => SELECT NUMERIC '0x0abc';
  ?column?
  2748
  (1 row)
  ```

### Examples

The following example casts three strings as NUMERICs:

```sql
=> SELECT NUMERIC '12.3e3', '12.3p10'::NUMERIC, CAST('0x12.3p-10e3' AS NUMERIC);
---------+---------+---------------
  12300 |  12595.2 |  17.76123046875000
 (1 row)
```

This example casts a VARBINARY string into a LONG VARBINARY data type:

```sql
=> SELECT B'101111000'::LONG VARBINARY;
?column?
---------
\001x
```
The following example concatenates a CHAR with a LONG VARCHAR, resulting in a LONG VARCHAR:

```sql
=> \set s `"``cat longfile.txt`'''
=> SELECT length ('a' || s :: LONG VARCHAR);
      length
    ---------
        65002
(1 row)
```

The following example casts a combination of NUMERIC and INTEGER data into a NUMERIC result:

```sql
=> SELECT (18. + 3./16)/1024*1000;
?column?
------------------------
 17.76123046875000000000
(1 row)
```

Note: In SQL expressions, pure numbers between \((-2^{63}-1)\) and \((2^{63}-1)\) are INTEGERS. Numbers with decimal points are NUMERIC.

See Also

- Data Type Coercion Chart
- Data Type Coercion Operators (CAST)

**Data Type Coercion Chart**

Conversion Types

The following table defines all possible type conversions that Vertica supports. The data types in the first column of the table are the inputs to convert, while data types listed across the second heading row indicate the resultant assignments.
<table>
<thead>
<tr>
<th>Data Types</th>
<th>Implicit</th>
<th>Explicit</th>
<th>Assignment without numeric meaning</th>
<th>Conversion without explicit casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td></td>
<td></td>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>BOOLEAN</td>
<td>NUMERIC</td>
<td>INTEGER</td>
<td>LONG VARCHAR</td>
</tr>
<tr>
<td></td>
<td>FLOAT</td>
<td>FLOAT</td>
<td>INTERVAL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>FLOAT</td>
<td></td>
<td>INTEGER</td>
<td>LONG VARCHAR</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td></td>
<td>INTEGER</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>FLOAT</td>
<td>CHAR</td>
<td>BOOLEAN</td>
<td>VOID</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
<td>CHAR</td>
<td>VARCHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>FLOAT</td>
<td>VARCHAR</td>
<td>BOOLEAN</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CHAR</td>
<td>FLOAT</td>
<td>VARCHAR</td>
<td>BOOLEAN</td>
<td>CHAR</td>
</tr>
</tbody>
</table>
## Conversion Types

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Implicit</th>
<th>Explicit</th>
<th>Assignment</th>
<th>Assignment without numeric meaning</th>
<th>Conversion without explicit casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMPZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIMETZ</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMETZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* CHAR length \( \geq 36 \)
Vertica supports data type conversion of values without explicit casting, such as `NUMERIC (10,6) -> NUMERIC(18,4)`. Implicit data type conversion occurs automatically when converting values of different, but compatible, types to the target column's data type. For example, when adding values, `(INTEGER + NUMERIC)`, the result is implicitly cast to a `NUMERIC` type to accommodate the prominent type in the statement. Depending on the input data types, different precision and scale can occur.

An explicit type conversion must occur when the source data cannot be cast implicitly to the target column's data type.

### Assignment Conversion

In data assignment conversion, coercion implicitly occurs when values are assigned to database columns in an `INSERT` or `UPDATE ... SET` statement. For example, in a statement that includes `INSERT ... VALUES('2.5')`, where the target column data type is `NUMERIC(18,5)`, a cast from `VARCHAR` to the column data type is inferred.

In an assignment without numeric meaning, the value is subject to `CHAR/VARCHAR/LONG VARCHAR` comparisons.
See Also

- Data Type Coercion
- Data Type Coercion Operators (CAST)
SQL Functions

Functions return information from the database. Except for Vertica-specific functions, you can use a function anywhere an expression is allowed.

This chapter describes each function that Vertica supports. The Behavior Type section of these descriptions categorizes the function's return behavior as one of the following:

Immutable (invariant)
When run with a given set of arguments, immutable functions such as \texttt{AVG()} always produce the same result, regardless of environment or session settings such as locale.

Some immutable functions can take an optional stable argument; in this case they are treated as Stable functions.

Stable
When run with a given set of arguments, stable functions produce the same result within a single query or scan operation. However, a stable function can produce different results when issued under different environments or at different times, such as change of locale and time zone—for example, \texttt{SYSDATE()} and 'today'.

Volatile
Regardless of their arguments or environment, volatile functions can return a different result with each invocation—for example, \texttt{UUID_GENERATE()}.

Note: All Vertica-specific functions are volatile; thus, the descriptions of these functions omit a Behavior Type section.
Aggregate Functions

Note: All functions in this section that have an analytic function counterpart are appended with [Aggregate] to avoid confusion between the two.

Aggregate functions summarize data over groups of rows from a query result set. The groups are specified using the GROUP BY clause. They are allowed only in the select list and in the HAVING and ORDER BY clauses of a SELECT statement (as described in Aggregate Expressions).

Notes

- Except for COUNT, these functions return a null value when no rows are selected. In particular, SUM of no rows returns NULL, not zero.

- In some cases you can replace an expression that includes multiple aggregates with an single aggregate of an expression. For example SUM(x) + SUM(y) can be expressed as as SUM(x+y) (where x and y are NOT NULL).

- Vertica does not support nested aggregate functions.

You can also use some of the simple aggregate functions as analytic (window) functions. See Analytic Functions for details. See also SQL Analytics in Analyzing Data.

APPROXIMATE_COUNT_DISTINCT

Returns the number of distinct non-NULL values in a data set.

Behavior Type

Immutable

Syntax

APPROXIMATE_COUNT_DISTINCT ( expression[, error-tolerance ] )
Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Value to be evaluated using any data type that supports equality comparison.</th>
</tr>
</thead>
<tbody>
<tr>
<td>error-tolerance</td>
<td>Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation. You can set error-tolerance to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation. Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance. If you omit this argument, Vertica uses an error tolerance of 1.75 (%).</td>
</tr>
</tbody>
</table>

Restrictions

APPROXIMATE_COUNT_DISTINCT and DISTINCT aggregates cannot be in the same query block.

Error Tolerance

APPROXIMATE_COUNT_DISTINCT(x, error-tolerance) returns a value equal to COUNT(DISTINCT x), with an error that is lognormally distributed with standard deviation.

Parameter error-tolerance is optional. Supply this argument to specify the desired standard deviation. error-tolerance is defined as 2.17 standard deviations, which corresponds to a 97 percent confidence interval:

\[
\text{standard-deviation} = \frac{\text{error tolerance}}{2.17}
\]

For example:

- **error-tolerance = 1**
  The default setting, corresponds to a standard deviation. 97 percent of the time, APPROXIMATE_COUNT_DISTINCT(x, 5) returns a value between:
- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) / 1.01 \)
- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) \times 1.01 \)

**error-tolerance** = 5

97 percent of the time, \( \text{APPROXIMATE}_{-} \text{COUNT}_{-} \text{DISTINCT} \left( x \right) \) returns a value between:

- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) / 1.05 \)
- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) \times 1.05 \)

A 99 percent confidence interval corresponds to 2.58 standard deviations. To set **error-tolerance** confidence level corresponding to 99 (instead of a 97) percent, multiply **error-tolerance** by \( 2.17 / 2.58 = 0.841 \).

For example, if you specify **error-tolerance** as \( 5 \times 0.841 = 4.2 \), \( \text{APPROXIMATE}_{-} \text{COUNT}_{-} \text{DISTINCT}(x,4.2) \) returns values 99 percent of the time between:

- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) \text{COUNT} / 1.05 \)
- \( \text{COUNT} \left( \text{DISTINCT} \ x \right) \times 1.05 \)

**Examples**

Count the total number of distinct values in column **product_key** from table **store.store_sales_fact**:

```sql
=> SELECT COUNT(DISTINCT product_key) FROM store.store_sales_fact;
COUNT
-------
19982
(1 row)
```

Count the approximate number of distinct values in **product_key** with various error tolerances. The smaller the error tolerance, the closer the approximation:

```sql
=> SELECT APPROXIMATE_COUNT_DISTINCT(product_key,5) AS five_pct_accuracy,
   APPROXIMATE_COUNT_DISTINCT(product_key,1) AS one_pct_accuracy,
   APPROXIMATE_COUNT_DISTINCT(product_key,.88) AS point_eighteight_pct_accuracy
FROM store.store_sales_fact;

<table>
<thead>
<tr>
<th>five_pct_accuracy</th>
<th>one_pct_accuracy</th>
<th>point_eighteight_pct_accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>19431</td>
<td>19921</td>
<td>19921</td>
</tr>
</tbody>
</table>
(1 row)
```
See Also

- APPROXIMATE_COUNT_DISTINCT_SYNOPSIS
- APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS
- COUNT [Aggregate]

**APPROXIMATE_COUNT_DISTINCT_SYNOPSIS**

Summarizes the information of distinct non-NULL values and materializes the result set in a VARBINARY or LONG VARBINARY synopsis object. The calculated result is within a specified range of error tolerance. You save the synopsis object in a Vertica table for use by APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS.

Note: If you are upgrading from a release to Vertica 8.1.1, you must drop the pre-existing synopsis object and recreate it to get the result.

**Behavior Type**

Immutable

**Syntax**

```
APPROXIMATE_COUNT_DISTINCT_SYNOPSIS ( expression[, error-tolerance] )
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>parameter</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>expression</strong></td>
<td>Value to evaluate using any data type that supports equality comparison.</td>
</tr>
<tr>
<td><strong>error-tolerance</strong></td>
<td>Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation. You can set error-tolerance to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation.</td>
</tr>
</tbody>
</table>
Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance.

If you omit this argument, Vertica uses an error tolerance of 1.75 (%).

For more details, see APPROXIMATE_COUNT_DISTINCT.

Restrictions

APPROXIMATE_COUNT_DISTINCT_SYNOPSIS and DISTINCT aggregates cannot be in the same query block.

Example

See APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS.

APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS

Calculates the number of distinct non-NULL values from the synopsis objects created by APPROXIMATE_COUNT_DISTINCT_SYNOPSIS.

Note: If you are upgrading from a release to Vertica 8.1.1, you must drop the pre-existing synopsis object and recreate it to get the result.

Behavior Type

Immutable

Syntax

APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS ( synopsis-obj[, error-tolerance ] )

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synopsis-obj</td>
<td>A synopsis object created by APPROXIMATE_COUNT_DISTINCT_SYNOPSIS.</td>
</tr>
</tbody>
</table>
**error-tolerance**

Numeric value that represents the desired percentage of error tolerance, distributed around the value returned by this function. The smaller the error tolerance, the closer the approximation.

You can set *error-tolerance* to a minimum value of 0.88, with an error than is lognormally distributed with standard deviation. Vertica imposes no maximum restriction, but any value greater than 5 is implemented with 5% error tolerance.

If you omit this argument, Vertica uses an error tolerance of 1.75 (%).

For more details, see [APPROXIMATE_COUNT_DISTINCT](#).

### Restrictions

[APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS](#) and DISTINCT aggregates cannot be in the same query block.

### Examples

The following examples review and compare different ways to obtain a count of unique values in a table column:

**Return an exact count of unique values in column product_key, from table store.store_sales_fact:**

```sql
=> \timing
Timing is on.
=> SELECT COUNT(DISTINCT product_key) from store.store_sales_fact;
   count
  -------
  19982
  (1 row)

Time: First fetch (1 row): 553.033 ms. All rows formatted: 553.075 ms
```

**Return an approximate count of unique values in column product_key:**

```sql
=> SELECT APPROXIMATE_COUNT_DISTINCT(product_key) as unique_product_keys
    FROM store.store_sales_fact;
   unique_product_keys
-----------------------
  19921
  (1 row)

Time: First fetch (1 row): 394.562 ms. All rows formatted: 394.600 ms
```
Create a synopsis object that represents a set of store.store_sales_fact data with unique product_key values, store the synopsis in the new table my_summary:

```sql
=> CREATE TABLE my_summary AS SELECT APPROXIMATE_COUNT_DISTINCT_SYNOPSIS (product_key) syn
   FROM store.store_sales_fact;
```

Time: First fetch (0 rows): 582.662 ms. All rows formatted: 582.682 ms

Return a count from the saved synopsis:

```sql
=> SELECT APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS(syn) FROM my_summary;
ApproxCountDistinctOfSynopsis
-------------------------------
  19921
(1 row)
```

Time: First fetch (1 row): 105.295 ms. All rows formatted: 105.335 ms

**APPROXIMATE_MEDIAN [Aggregate]**

Computes the approximate median of an expression over a group of rows. The function returns a FLOAT value.

APPROXIMATE_MEDIAN is an alias of APPROXIMATE_PERCENTILE [Aggregate] with a parameter of 0.5.

Note: This function is best suited for large groups of data. If you have a small group of data, use the exact MEDIAN [Analytic] function.

**Behavior Type**

Immutable

**Syntax**

```
APPROXIMATE_MEDIAN ( expression )
```

**Parameters**

<table>
<thead>
<tr>
<th>expression</th>
<th>Any FLOAT or INTEGER data type. The function returns the approximate middle value or an interpolated value that would be the approximate</th>
</tr>
</thead>
</table>
middle value once the values are sorted. Null values are ignored in the calculation.

Examples

Tip: For optimal performance when using GROUP BY in your query, verify that your table is sorted on the GROUP BY column.

The following examples uses this table:

```sql
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT) ORDER BY state;
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;
```

Calculate the approximate median of all sales in this table:

```sql
=> SELECT APPROXIMATE_MEDIAN (sales) FROM allsales;
APPROXIMATE_MEDIAN
---------------------
    20
(1 row)
```

Modify the query to group sales by state, and obtain the approximate median for each one:

```sql
=> SELECT state, APPROXIMATE_MEDIAN(sales) FROM allsales GROUP BY state;
state | APPROXIMATE_MEDIAN
------|---------------------
MA    |        35
NY    |        20
(2 rows)
```

See Also

- `MEDIAN [Analytic]`
- `PERCENTILE_CONT [Analytic]`
APPROXIMATE_PERCENTILE [Aggregate]

Computes the approximate percentile of an expression over a group of rows. This function returns a FLOAT value. Nulls are ignored.

**Note**: This function is best suited for large groups of data. If you have a small group of data, use the exact PERCENTILE_CONT [Analytic] function.

**Behavior Type**

Immutable

**Syntax**

```
APPROXIMATE_PERCENTILE ( expression USING PARAMETERS percentile = number )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression</code></td>
<td>Any FLOAT or INTEGER data type. Null values are ignored.</td>
</tr>
<tr>
<td><code>number</code></td>
<td>Percentile value, which must be a FLOAT constant ranging from 0 to 1 (inclusive).</td>
</tr>
</tbody>
</table>

**Examples**

**Tip**: For optimal performance when using GROUP BY in your query, verify that your table is sorted on the GROUP BY column.

The following example uses this table:
CREATE TABLE allsales(state VARCHAR(20), name VARCHAR(20), sales INT) ORDER BY state;
INSERT INTO allsales VALUES('MA', 'A', 60);
INSERT INTO allsales VALUES('NY', 'B', 20);
INSERT INTO allsales VALUES('NY', 'C', 15);
INSERT INTO allsales VALUES('MA', 'D', 20);
INSERT INTO allsales VALUES('MA', 'E', 50);
INSERT INTO allsales VALUES('NY', 'F', 40);
INSERT INTO allsales VALUES('MA', 'G', 10);
COMMIT;

Calculate the approximate percentile for sales in each state:

```sql
=> SELECT state, APPROXIMATE_PERCENTILE(sales USING PARAMETERS percentile=0.5) AS median FROM allsales GROUP BY state;
state | median
-------+-------
MA     | 35
NY     | 20
(2 rows)
```

See Also

- MEDIAN [Analytic]
- PERCENTILE_CONT [Analytic]
- SQL Analytics

AVG [Aggregate]

Computes the average (arithmetic mean) of an expression over a group of rows. AVG always returns a DOUBLE PRECISION value.

The AVG aggregate function differs from the AVG analytic function, which computes the average of an expression over a group of rows within a window.

Behavior Type

Immutable

Syntax

`AVG ( [ ALL | DISTINCT ] expression )`
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Invokes the aggregate function for all rows in the group (default).</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Invokes the aggregate function for all distinct non-null values of the expression found in the group.</td>
</tr>
</tbody>
</table>

- **expression**
  - The value whose average is calculated over a set of rows, any expression that can have a DOUBLE PRECISION result.

### Overflow Handling

By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see [Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG](#).

### Examples

The following query returns the average income from the customer table:

```sql
=> SELECT AVG(annual_income) FROM customer_dimension;
    AVG
---------------
 2104270.6485
(1 row)
```

### See Also

- [COUNT [Aggregate]](#)
- [SUM [Aggregate]](#)
- [Numeric Data Types](#)
BIT_AND

Takes the bitwise AND of all non-null input values. If the input parameter is NULL, the return value is also NULL.

Behavior Type

Immutable

Syntax

BIT_AND ( expression )

Parameters

| expression | The BINARY or VARBINARY input value to evaluate. BIT_AND operates on VARBINARY types explicitly and on BINARY types implicitly through casts. |

Returns

BIT_AND returns:

- The same value as the argument data type.
- 1 for each bit compared, if all bits are 1; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, BIT_AND ignores the null value and extends the value f to f0.

Example

The example that follows uses table t with a single column of VARBINARY data type:

```sql
=> CREATE TABLE t ( c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
```
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));

Query table t to see column c output:

```sql
=> SELECT TO_HEX(c) FROM t;
<table>
<thead>
<tr>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>ff00</td>
</tr>
<tr>
<td>ffff</td>
</tr>
<tr>
<td>f00f</td>
</tr>
</tbody>
</table>
(3 rows)
```

Query table t to get the AND value for column c:

```sql
=> SELECT TO_HEX(BIT_AND(c)) FROM t;
<table>
<thead>
<tr>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>f000</td>
</tr>
</tbody>
</table>
(1 row)
```

The function is applied pairwise to all values in the group, resulting in f000, which is determined as follows:

1. ff00 (record 1) is compared with ffff (record 2), which results in ff00.
2. The result from the previous comparison is compared with f00f (record 3), which results in f000.

See Also

Binary Data Types

**BIT_OR**

Takes the bitwise OR of all non-null input values. If the input parameter is NULL, the return value is also NULL.

**Behavior Type**

Immutable
Syntax

BIT_OR ( expression )

Parameters

| expression | The [BINARY | VARBINARY] input value to be evaluated. BIT_OR() operates on VARBINARY types explicitly and on BINARY types implicitly through casts. |

Returns

BIT_OR returns:

- The same value as the argument data type.
- 1 for each bit compared, if any bit is 1; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, the function ignores the null value and extends the value f to f0.

Example

The example that follows uses table t with a single column of VARBINARY data type:

```sql
=> CREATE TABLE t ( c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table t to see column c output:

```sql
=> SELECT TO_HEX(c) FROM t;
TO_HEX
--------
ff00
ffff
f00f
(3 rows)
```

Query table t to get the OR value for column c:
The function is applied pairwise to all values in the group, resulting in \texttt{ffff}, which is determined as follows:

1. \texttt{ff00} (record 1) is compared with \texttt{ffff}, which results in \texttt{ffff}.

2. The \texttt{ff00} result from the previous comparison is compared with \texttt{f00f} (record 3), which results in \texttt{ffff}.

**See Also**

Binary Data Types

**BIT\_XOR**

Takes the bitwise XOR of all non-null input values. If the input parameter is NULL, the return value is also NULL.

**Behavior Type**

Immutable

**Syntax**

\[
\text{BIT\_XOR ( expression )}
\]

**Parameters**

<table>
<thead>
<tr>
<th>\texttt{expression}</th>
<th>The BINARY or VARBINARY input value to evaluate. BIT_XOR operates on VARBINARY types explicitly and on BINARY types implicitly through \texttt{casts}.</th>
</tr>
</thead>
</table>
Returns

BIT_XOR returns:

- The same value as the argument data type.
- 1 for each bit compared, if there are an odd number of arguments with set bits; otherwise 0.

If the columns are different lengths, the return values are treated as though they are all equal in length and are right-extended with zero bytes. For example, given a group containing hex values ff, null, and f, the function ignores the null value and extends the value f to f0.

Example

First create a sample table and projections with binary columns:

The example that follows uses table t with a single column of VARBINARY data type:

```
=> CREATE TABLE t ( c VARBINARY(2) );
=> INSERT INTO t values(HEX_TO_BINARY('0xFF00'));
=> INSERT INTO t values(HEX_TO_BINARY('0xFFFF'));
=> INSERT INTO t values(HEX_TO_BINARY('0xF00F'));
```

Query table t to see column c output:

```
=> SELECT TO_HEX(c) FROM t;
   TO_HEX
   ----
   ff00
   ffff
   f00f
   (3 rows)
```

Query table t to get the XOR value for column c:

```
=> SELECT TO_HEX(BIT_XOR(c)) FROM t;
   TO_HEX
   ------
   f0f0
   (1 row)
```

See Also

Binary Data Types
BOOL_AND [Aggregate]

Processes Boolean values and returns a Boolean value result. If all input values are true, BOOL_AND returns \( \text{true} \). Otherwise it returns \( \text{false} \).

**Behavior Type**

Immutable

**Syntax**

BOOL_AND ( expression )

**Parameters**

| expression | A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type. |

**Examples**

The following example shows how to use aggregate functions BOOL_AND, BOOL_OR, and BOOL_XOR. The sample table mixers includes columns for models and colors.

```
=> CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));
CREATE TABLE
```

Insert sample data into the table. The sample adds two color fields for each model.

```
=> INSERT INTO mixers
    SELECT 'beginner', 'green'
    UNION ALL
    SELECT 'intermediate', 'blue'
    UNION ALL
    SELECT 'intermediate', 'blue'
    UNION ALL
    SELECT 'advanced', 'green'
    UNION ALL
    SELECT 'advanced', 'blue'
    UNION ALL
    SELECT 'professional', 'blue'
```
```
UNION ALL
SELECT 'professional', 'green'
UNION ALL
SELECT 'beginner', 'green';
```

Query the table. The result shows models that have two blue (BOOL_AND), one or two blue (BOOL_OR), and specifically not more than one blue (BOOL_XOR) mixer.

```
=> SELECT model,
    BOOL_AND(colors= 'blue')AS two_blue,
    BOOL_OR(colors= 'blue')AS one_or_two_blue,
    BOOL_XOR(colors= 'blue')AS specifically_not_more_than_one_blue
FROM mixers
GROUP BY model;
```

<table>
<thead>
<tr>
<th>model</th>
<th>two_blue</th>
<th>one_or_two_blue</th>
<th>specifically_not_more_than_one_blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>advanced</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>beginner</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>intermediate</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>professional</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

(4 rows)

See Also

- BOOL_AND [Analytic]
- BOOL_OR [Aggregate]
- BOOL_XOR [Aggregate]
- Boolean Data Type

**BOOL_OR [Aggregate]**

Processes Boolean values and returns a Boolean value result. If at least one input value is true, BOOL_OR returns t. Otherwise, it returns f.

**Behavior Type**

Immutable
Syntax

BOOL_OR ( expression )

Parameters

| expression | A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type. |

Examples

The following example shows how to use aggregate functions BOOL_AND, BOOL_OR, and BOOL_XOR. The sample table mixers includes columns for models and colors.

```sql
=> CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));
CREATE TABLE

=> INSERT INTO mixers
   SELECT 'beginner', 'green'
   UNION ALL
   SELECT 'intermediate', 'blue'
   UNION ALL
   SELECT 'intermediate', 'blue'
   UNION ALL
   SELECT 'advanced', 'green'
   UNION ALL
   SELECT 'advanced', 'blue'
   UNION ALL
   SELECT 'professional', 'blue'
   UNION ALL
   SELECT 'professional', 'green'
   UNION ALL
   SELECT 'beginner', 'green';
OUTPUT
-------
8 (1 row)

Query the table. The result shows models that have two blue (BOOL_AND), one or two blue (BOOL_OR), and specifically not more than one blue (BOOL_XOR) mixer.

```sql
=> SELECT model,
    BOOL_AND(colors= 'blue')AS two_blue,
    BOOL_OR(colors= 'blue')AS one_or_two_blue,
```
BOOL_XOR(colors= 'blue') AS specifically_not_more_than_one_blue
FROM mixers
GROUP BY model;

<table>
<thead>
<tr>
<th></th>
<th>two_blue</th>
<th>one_or_two_blue</th>
<th>specifically_not_more_than_one_blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>advanced</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>beginner</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>intermediate</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>professional</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

(4 rows)

See Also

- BOOL_OR [Analytic]
- BOOL_AND [Aggregate]
- BOOL_XOR [Aggregate]
- Boolean Data Type

**BOOL_XOR [Aggregate]**

Processes Boolean values and returns a Boolean value result. If specifically only one input value is true, BOOL_XOR returns t. Otherwise, it returns f.

**Behavior Type**

Immutable

**Syntax**

BOOL_XOR ( expression )

**Parameters**

| expression | A Boolean data type or any non-Boolean data type that can be implicitly coerced to a Boolean data type. |
Examples

The following example shows how to use aggregate functions BOOL_AND, BOOL_OR, and BOOL_XOR. The sample table `mixers` includes columns for models and colors.

```sql
CREATE TABLE mixers(model VARCHAR(20), colors VARCHAR(20));
```

Insert sample data into the table. The sample adds two color fields for each model.

```sql
=> INSERT INTO mixers
SELECT 'beginner', 'green'
UNION ALL
SELECT 'intermediate', 'blue'
UNION ALL
SELECT 'intermediate', 'blue'
UNION ALL
SELECT 'advanced', 'green'
UNION ALL
SELECT 'advanced', 'blue'
UNION ALL
SELECT 'professional', 'blue'
UNION ALL
SELECT 'professional', 'green'
UNION ALL
SELECT 'beginner', 'green';
```

Query the table. The result shows models that have two blue (BOOL_AND), one or two blue (BOOL_OR), and specifically not more than one blue (BOOL_XOR) mixer.

```sql
=> SELECT model,
   BOOL_AND(colors = 'blue')AS two_blue,
   BOOL_OR(colors = 'blue')AS one_or_two_blue,
   BOOL_XOR(colors = 'blue')AS specifically_not_more_than_one_blue
FROM mixers
GROUP BY model;
```

```
<table>
<thead>
<tr>
<th>model</th>
<th>two_blue</th>
<th>one_or_two_blue</th>
<th>specifically_not_more_than_one_blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>advanced</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>beginner</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>intermediate</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>professional</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>
```

(4 rows)
See Also

- BOOL_XOR [Analytic]
- BOOL_AND [Aggregate]
- BOOL_OR [Aggregate]
- Boolean Data Type

CORR

Returns the DOUBLE PRECISION coefficient of correlation of a set of expression pairs. CORR eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

Syntax

CORR ( expression1, expression2 )

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

Example

=> SELECT CORR (Annual_salary, Employee_age) FROM employee_dimension;
   CORR
  ------------------------
  -0.00719153413192422
  (1 row)
COUNT [Aggregate]

Returns as a BIGINT the number of rows in each group where the expression is not NULL. If the query has no GROUP BY clause, COUNT returns the number of table rows.

The COUNT aggregate function differs from the \textit{COUNT} analytic function, which returns the number over a group of rows within a window.

Behavior Type

Immutable

Syntax

\texttt{COUNT ([ * ] [ ALL | DISTINCT ] \textit{expression} )}

Parameters

<table>
<thead>
<tr>
<th>*</th>
<th>Specifies to count all rows in the specified table or each group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>DISTINCT</td>
</tr>
<tr>
<td></td>
<td>\begin{itemize} \item ALL (default): Counts all rows where \textit{expression} evaluates to a non-null value. \item DISTINCT: Counts all rows where \textit{expression} evaluates to a distinct non-null value. \end{itemize}</td>
</tr>
<tr>
<td>\textit{expression}</td>
<td>The column or expression whose non-null values are counted.</td>
</tr>
</tbody>
</table>

Examples

The following query returns the number of distinct values in the primary\textunderscore key column of the date\textunderscore dimension table:

\begin{verbatim}
=> SELECT COUNT (DISTINCT date_key) FROM date_dimension;
\end{verbatim}
This example returns all distinct values of evaluating the expression \(x+y\) for all inventory_fact records.

\[
=> \text{SELECT COUNT (DISTINCT date_key + product_key) FROM inventory_fact;}
\]

<table>
<thead>
<tr>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1826</td>
</tr>
</tbody>
</table>

You can create an equivalent query using the LIMIT keyword to restrict the number of rows returned:

\[
=> \text{SELECT COUNT(date_key + product_key) FROM inventory_fact GROUP BY date_key LIMIT 10;}
\]

```
<table>
<thead>
<tr>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>21560</td>
</tr>
</tbody>
</table>
```

This query returns the number of distinct values of date_key in all records with the specific distinct product_key value.

\[
=> \text{SELECT product_key, COUNT (DISTINCT date_key) FROM inventory_fact GROUP BY product_key LIMIT 10;}
\]

```
<table>
<thead>
<tr>
<th>product_key</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
```

This query counts each distinct product_key value in inventory_fact table with the constant 1.
This query selects each distinct \texttt{date\_key} value and counts the number of distinct \texttt{product\_key} values for all records with the specific \texttt{product\_key} value. It then sums the \texttt{qty\_in\_stock} values in all records with the specific \texttt{product\_key} value and groups the results by \texttt{date\_key}.

This query selects each distinct \texttt{product\_key} value and then counts the number of distinct \texttt{date\_key} values for all records with the specific \texttt{product\_key} value. It also counts the number of distinct \texttt{warehouse\_key} values in all records with the specific \texttt{product\_key} value.
This query selects each distinct product_key value, counts the number of distinct date_key and warehouse_key values for all records with the specific product_key value, and then sums all qty_in_stock values in records with the specific product_key value. It then returns the number of product_version values in records with the specific product_key value.

```sql
=> SELECT product_key, COUNT(DISTINCT date_key),
    COUNT(DISTINCT warehouse_key),
    SUM(qty_in_stock),
    COUNT(product_version)
FROM inventory_fact GROUP BY product_key LIMIT 15;
```

<table>
<thead>
<tr>
<th>product_key</th>
<th>count</th>
<th>count</th>
<th>sum</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>12</td>
<td>5530</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>18</td>
<td>9605</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>12</td>
<td>8404</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>18</td>
<td>10006</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>9</td>
<td>4794</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>13</td>
<td>7359</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>13</td>
<td>7828</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>15</td>
<td>9074</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>14</td>
<td>7032</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>12</td>
<td>5359</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>6049</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>12</td>
<td>6075</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>7</td>
<td>3470</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>13</td>
<td>5125</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>17</td>
<td>9277</td>
<td>18</td>
</tr>
</tbody>
</table>

(15 rows)

The following example returns the number of warehouses from the warehouse dimension table:

```sql
=> SELECT COUNT(warehouse_name) FROM warehouse_dimension;

COUNT
------
100

(1 row)
```

This next example returns the total number of vendors:

```sql
=> SELECT COUNT(*) FROM vendor_dimension;

COUNT(*)
--------
100

(1 row)
```
See Also

- Analytic Functions
- AVG [Aggregate]
- SUM [Aggregate]
- SQL Analytics
- APPROXIMATE_COUNT_DISTINCT
- APPROXIMATE_COUNT_DISTINCT_SYNOPSIS
- APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS

COVAR_POP

Returns the population covariance for a set of expression pairs. The return value is of type DOUBLE  PRECISION. COVAR_POP eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

Syntax

```
SELECT COVAR_POP( expression1, expression2 )
```

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE  PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE  PRECISION expression</td>
</tr>
</tbody>
</table>
Example

```sql
=> SELECT COVAR_POP (Annual_salary, Employee_age)
    FROM employee_dimension;

COVAR_POP
-----------------------
-9032.34810730019
(1 row)
```

**COVAR_SAMP**

Returns the sample covariance for a set of expression pairs. The return value is of type DOUBLE PRECISION. COVAR_SAMP eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.

**Syntax**

```sql
SELECT COVAR_SAMP ( expression1, expression2 )
```

**Parameters**

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

**Example**

```sql
=> SELECT COVAR_SAMP (Annual_salary, Employee_age)
    FROM employee_dimension;

COVAR_SAMP
-----------------------
-9033.25143244343
(1 row)
```
GROUP_ID

Uniquely identifies duplicate sets for GROUP BY queries that return duplicate grouping sets. This function returns one or more integers, starting with zero (0), as identifiers.

For the number of duplicates $n$ for a particular grouping, GROUP_ID returns a range of sequential numbers, 0 to $n-1$. For the first each unique group it encounters, GROUP_ID returns the value 0. If GROUP_ID finds the same grouping again, the function returns 1, then returns 2 for the next found grouping, and so on.

Note: Use GROUP_ID only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.

Behavior Type

Immutable

Syntax

GROUP_ID ()

Examples

This example shows how GROUP_ID creates unique identifiers when a query produces duplicate groupings. For an expenses table, the following query groups the results by category of expense and year and rolls up the sum for those two columns. The results have duplicate groupings for category and NULL. The first grouping has a GROUP_ID of 0, and the second grouping has a GROUP_ID of 1.

```
=> SELECT Category, Year, SUM(Amount), GROUPING_ID(Category, Year),
    GROUP_ID() FROM expenses GROUP BY Category, ROLLUP(Category,Year)
ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING_ID</th>
<th>GROUP_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
See Also

- CUBE Aggregate
- GROUPING
- GROUPING_ID
- GROUPING SETS Aggregate
- GROUP BY Clause
- ROLLUP Aggregate

GROUPING

Disambiguates the use of NULL values when GROUP BY queries with multilevel aggregates generate NULL values to identify subtotals in grouping columns. Such NULL values from the original data can also occur in rows. GROUPING returns 1, if the value of expression is:

- NULL, representing an aggregated value
- 0 for any other value, including NULL values in rows

Note: Use GROUPING only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.

Behavior Type

Immutable

Syntax

GROUPING ( expression )
Parameters

| expression | An expression in the GROUP BY clause |

Examples

The following query uses the GROUPING function, taking one of the GROUP BY expressions as an argument. For each row, GROUPING returns one of the following:

- **0**: The column is part of the group for that row
- **1**: The column is not part of the group for that row

The 1 in the GROUPING(Year) column for electricity and books indicates that these values are subtotals. The right-most column values for both GROUPING(Category) and GROUPING(Year) are 1. This value indicates that neither column contributed to the GROUP BY. The final row represents the total sales.

```sql
=> SELECT Category, Year, SUM(Amount),
   GROUPING(Category), GROUPING(Year) FROM expenses
   GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

- **CUBE Aggregate**
- **GROUP_ID**
- **GROUPING_ID**
- **GROUPING SETS Aggregate**
GROUPING_ID

Concatenates the set of Boolean values generated by the GROUPING function into a bit vector. GROUPING_ID treats the bit vector as a binary number and returns it as a base-10 value that identifies the grouping set combination.

By using GROUPING_ID you avoid the need for multiple, individual GROUPING functions. GROUPING_ID simplifies row-filtering conditions, because rows of interest are identified using a single return from GROUPING_ID = n. Use GROUPING_ID to identify grouping combinations.

Note: Use GROUPING_ID only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.

Behavior Type

Immutable

Syntax

GROUPING_ID ( [expression[,...] ] )

expression | An expression that matches one of the expressions in the GROUP BY clause. If the GROUP BY clause includes a list of expressions, GROUPING_ID returns a number corresponding to the GROUPING bit vector associated with a row.

Examples

This example shows how calling GROUPING_ID without an expression returns the GROUPING bit vector associated with a full set of multilevel aggregate expressions. The GROUPING_ID
value is comparable to GROUPING_ID(a, b) because GROUPING_ID() includes all columns in the GROUP BY ROLLUP:

```sql
=> SELECT a,b,COUNT(*), GROUPING_ID() FROM T GROUP BY ROLLUP(a,b);
```

In the following query, the GROUPING(Category) and GROUPING(Year) columns have three combinations:

- 0,0
- 0,1
- 1,1

```sql
=> SELECT Category, Year, SUM(Amount),
    GROUPING(Category), GROUPING(Year) FROM expenses
GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

GROUPING_ID converts these values as follows:

<table>
<thead>
<tr>
<th>Binary Set Values</th>
<th>Decimal Equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>Category, Year</td>
</tr>
</tbody>
</table>

The following query returns the single number for each GROUP BY level that appears in the gr_id column:

```sql
=> SELECT Category, Year, SUM(Amount),
    GROUPING(Category), GROUPING(Year), GROUPING_ID(Category,Year) AS gr_id
FROM expenses GROUP BY ROLLUP(Category, Year);
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
<th>gr_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The `gr_id` value determines the GROUP BY level for each row:

<table>
<thead>
<tr>
<th>GROUP BY Level</th>
<th>GROUP BY Row Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Total sum</td>
</tr>
<tr>
<td>1</td>
<td>Category</td>
</tr>
<tr>
<td>0</td>
<td>Category, year</td>
</tr>
</tbody>
</table>

You can also use the `DECODE` function to give the values more meaning by comparing each search value individually:

```sql
=> SELECT Category, Year, SUM(AMOUNT), DECODE(GROUPING_ID(Category, Year),
   3, 'Total',
   1, 'Category',
   0, 'Category,Year')
   AS GROUP_NAME FROM expenses GROUP BY ROLLUP(Category, Year);
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUP_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>99.96</td>
<td>Category</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>449.96</td>
<td>Category</td>
</tr>
<tr>
<td></td>
<td>549.92</td>
<td>1</td>
<td>Total</td>
</tr>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>Category,Year</td>
</tr>
</tbody>
</table>

See Also

- CUBE Aggregate
- GROUP_ID
- GROUPING
- GROUPING SETS Aggregate
GROUP BY Clause

ROLLUP Aggregate

MAX [Aggregate]

Returns the greatest value of an expression over a group of rows. The return value has the same type as the expression data type.

The MAX analytic function function differs from the aggregate function, in that it returns the maximum value of an expression over a group of rows within a window.

Aggregate functions MIN and MAX can operate with Boolean values. MAX can act upon a Boolean data type or a value that can be implicitly converted to a Boolean. If at least one input value is true, MAX returns t (true). Otherwise, it returns f (false). In the same scenario, MIN returns t (true) if all input values are true. Otherwise it returns f.

Behavior Type

Immutable

Syntax

MAX ( expression )

Parameters

| expression | Any expression for which the maximum value is calculated, typically a column reference. |

Examples

The following query returns the largest value in column sales_dollar_amount.

```sql
=> SELECT MAX(sales_dollar_amount) AS highest_sale FROM store.store_sales_fact;

<table>
<thead>
<tr>
<th>highest_sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
</tr>
</tbody>
</table>

(1 row)```
The following example shows you the difference between the \texttt{MIN} and \texttt{MAX} aggregate functions when you use them with a Boolean value. The sample creates a table, adds two rows of data, and shows sample output for \texttt{MIN} and \texttt{MAX}.

```sql
=> CREATE TABLE min_max_functions (torf BOOL);

=> INSERT INTO min_max_functions VALUES (1);
=> INSERT INTO min_max_functions VALUES (0);

=> SELECT * FROM min_max_functions;
torf
------
t f
(2 rows)

=> SELECT min(torf) FROM min_max_functions;
min
------
f  
(1 row)

=> SELECT max(torf) FROM min_max_functions;
max
------
t  
(1 row)
```

See Also

- Data Aggregation

\textbf{MIN [Aggregate]}

Returns the smallest value of an expression over a group of rows. The return value has the same type as the expression data type.

The \texttt{MIN} \texttt{analytic function} differs from the aggregate function, in that it returns the minimum value of an expression over a group of rows within a window.

Aggregate functions \texttt{MIN} and \texttt{MAX} can operate with Boolean values. \texttt{MAX} can act upon a \texttt{Boolean data type} or a value that can be implicitly converted to a Boolean. If at least one input value is true, \texttt{MAX} returns \texttt{t} (true). Otherwise, it returns \texttt{f} (false). In the same scenario, \texttt{MIN} returns \texttt{t} (true) if all input values are true. Otherwise it returns \texttt{f}.

\textbf{Behavior Type}

Immutable
Syntax

MIN ( expression )

Parameters

| expression | Any expression for which the minimum value is calculated, typically a column reference. |

Examples

The following query returns the lowest salary from the employee dimension table.

This example shows how you can query to return the lowest salary from the employee dimension table.

```sql
=> SELECT MIN(annual_salary) AS lowest_paid FROM employee_dimension;
 lowest_paid
--------
 1200
(1 row)
```

The following example shows you the difference between the MIN and MAX aggregate functions when you use them with a Boolean value. The sample creates a table, adds two rows of data, and shows sample output for MIN and MAX.

```sql
=> CREATE TABLE min_max_functions (torf BOOL);
=> INSERT INTO min_max_functions VALUES (1);
=> INSERT INTO min_max_functions VALUES (0);
=> SELECT * FROM min_max_functions;
 torf
 ----
 t
 f
(2 rows)

=> SELECT min(torf) FROM min_max_functions;
 min
----
 f
(1 row)

=> SELECT max(torf) FROM min_max_functions;
 max
----
"
See Also

Data Aggregation

REGR_AVGX

Returns the DOUBLE PRECISION average of the independent expression in an expression pair. REGR_AVGX eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_AVGX returns NULL.

Syntax

SELECT REGR_AVGX ( expression1, expression2 )

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT REGR_AVGX (Annual_salary, Employee_age)
       FROM employee_dimension;
REGR_AVGX
--------
 39.321
(1 row)
```

REGR_AVGY

Returns the DOUBLE PRECISION average of the dependent expression in an expression pair. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns NULL.
Syntax

REGR_AVGY (expression1, expression2)

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

Example

=> SELECT REGR_AVGY (Annual_salary, Employee_age)
   FROM employee_dimension;
REGR_AVGY
----------
58354.4913
(1 row)

REGR_COUNT

Returns the count of all rows in an expression pair. The function eliminates expression pairs where either expression in the pair is NULL. If no rows remain, the function returns 0.

Syntax

SELECT REGR_COUNT (expression1, expression2)

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>
Example

```sql
=> SELECT REGR_COUNT (Annual_salary, Employee_age) FROM employee_dimension;
REGR_COUNT
----------
  10000
(1 row)
```

REGR_INTERCEPT

Returns the y-intercept of the regression line determined by a set of expression pairs. The return value is of type DOUBLE  PRECISION. REGR_INTERCEPT eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_INTERCEPT returns NULL.

Syntax

```sql
SELECT REGR_INTERCEPT ( expression1, expression2 )
```

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE  PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE  PRECISION expression</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT REGR_INTERCEPT (Annual_salary, Employee_age) FROM employee_dimension;
REGR_INTERCEPT
-------------
  59929.5490163437
(1 row)
```

REGR_R2

Returns the square of the correlation coefficient of a set of expression pairs. The return value is of type DOUBLE  PRECISION. REGR_R2 eliminates expression pairs where either expression in
the pair is NULL. If no rows remain, REGR_R2 returns NULL.

Syntax

SELECT REGR_R2 (expression1, expression2)

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT REGR_R2 (Annual_salary, Employee_age) FROM employee_dimension;
REGR_R2
5.17181631706311e-05
(1 row)
```

REGR_SLOPE

Returns the slope of the regression line, determined by a set of expression pairs. The return value is of type DOUBLE PRECISION. REGR_SLOPE eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_SLOPE returns NULL.

Syntax

SELECT REGR_SLOPE (expression1, expression2)

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>


Example

```sql
=> SELECT REGR_SLOPE (Annual_salary, Employee_age) FROM employee_dimension;
REGR_SLOPE
---------
-40.056400303749
(1 row)
```

**REGR_SXX**

Returns the sum of squares of the difference between the independent expression `expression2` and its average.

That is, `REGR_SXX` returns: 

\[ \sum [(expression2 - \text{average}(expression2))(expression2 - \text{average}(expression2))] \]

The return value is of type DOUBLE PRECISION. `REGR_SXX` eliminates expression pairs where either expression in the pair is NULL. If no rows remain, `REGR_SXX` returns NULL.

**Syntax**

```sql
SELECT REGR_SXX (expression1, expression2)
```

**Parameters**

<table>
<thead>
<tr>
<th><code>expression1</code></th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression2</code></td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

**Example**

```sql
=> SELECT REGR_SXX (Annual_salary, Employee_age) FROM employee_dimension;
REGR_SXX
---------
2254907.59
(1 row)
```
**REGR_SXY**

Returns the sum of products of the difference between the dependent expression \((expression1)\) and its average and the difference between the independent expression \((expression2)\) and its average.

That is, \(\text{REGR}_SXY\) returns: \(\Sigma[(expression1 - \text{average}(expression1))(expression2 - \text{average}(expression2))]\)

The return value is of type DOUBLE PRECISION. \(\text{REGR}_SXY\) eliminates expression pairs where either expression in the pair is NULL. If no rows remain, \(\text{REGR}_SXY\) returns NULL.

**Syntax**

```
SELECT REGR_SXY (expression1, expression2)
```

**Parameters**

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

**Example**

```
=> SELECT REGR_SXY (Annual_salary, Employee_age) FROM employee_dimension;
   REGR_SXY

-90323481.0730019

(1 row)
```

**REGR_SYY**

Returns the sum of squares of the difference between the dependent expression \((expression1)\) and its average.

That is, \(\text{REGR}_SYY\) returns: \(\Sigma[(expression1 - \text{average}(expression1))(expression1 - \text{average}(expression1))]\)
The return value is of type DOUBLE PRECISION. REGR_SYY eliminates expression pairs where either expression in the pair is NULL. If no rows remain, REGR_SYY returns NULL.

Syntax

```
SELECT REGR_SYY ( expression1, expression2 )
```

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>The dependent DOUBLE PRECISION expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>The independent DOUBLE PRECISION expression</td>
</tr>
</tbody>
</table>

Example

```
=> SELECT REGR_SYY (Annual_salary, Employee_age) FROM employee_dimension;

  REGR_SYY

--------------
69956728794707.2

(1 row)
```

STDDEV [Aggregate]

Evaluates the statistical sample standard deviation for each member of the group. The return value is the same as the square root of VAR_SAMP:

```
STDDEV(expression) = SQRT(VAR_SAMP(expression))
```

Behavior Type

Immutable

Syntax

```
STDDEV ( expression )
```
Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. STDDEV returns the same data type as expression. |

Related Functions

- Nonstandard function STDDEV is provided for compatibility with other databases. It is semantically identical to STDDEV_SAMP.

- This aggregate function differs from analytic function STDDEV, which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.

- When VAR_SAMP returns NULL, STDDEV returns NULL.

Examples

The following example returns the statistical sample standard deviation for each household ID from the customer_dimension table of the VMart example database:

```sql
=> SELECT STDDEV(household_id) FROM customer_dimension;
STDDEV
----------
8651.5084240071
```

STDDEV_POP [Aggregate]

Evaluates the statistical population standard deviation for each member of the group.

Behavior Type

Immutable
Syntax

STDDEV_POP ( expression )

Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. STDDEV_POP returns the same data type as expression. |

Related Functions

- This function differs from the analytic function STDDEV_POP, which evaluates the statistical population standard deviation for each member of the group of rows within a window.
- STDDEV_POP returns the same value as the square root of VAR_POP:
  
  \[
  \text{STDDEV\_POP}(\text{expression}) = \sqrt{\text{VAR\_POP}(\text{expression})}
  \]
- When VAR_SAMP returns NULL, this function returns NULL.

Examples

The following example returns the statistical population standard deviation for each household ID in the customer table.

```sql
=> SELECT STDDEV_POP(household_id) FROM customer_dimension;

STDDEV_POP

------------------------
 8651.41895973367

(1 row)
```

See Also

- Analytic Functions
- SQL Analytics
STDDEV_SAMP [Aggregate]

Evaluates the statistical sample standard deviation for each member of the group. The return value is the same as the square root of VAR_SAMP:

\[
\text{STDDEV}_\text{SAMP}(\text{expression}) = \text{SQRT}(\text{VAR}_\text{SAMP}(\text{expression}))
\]

Behavior Type

Immutable

Syntax

\[
\text{STDDEV}_\text{SAMP}(\text{expression})
\]

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. STDDEV_SAMP returns the same data type as expression.</td>
</tr>
</tbody>
</table>

Related Functions

- STDDEV_SAMP is semantically identical to nonstandard function STDDEV, which is provided for compatibility with other databases.

- This aggregate function differs from analytic function STDDEV_SAMP, which computes the statistical sample standard deviation of the current row with respect to the group of rows within a window.

- When VAR_SAMP returns NULL, STDDEV_SAMP returns NULL.

Examples

The following example returns the statistical sample standard deviation for each household ID from the customer dimension table.
=> SELECT STDDEV_SAMP(household_id) FROM customer_dimension;

<table>
<thead>
<tr>
<th>stddev_samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>8651.50842400771</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

**SUM [Aggregate]**

Computes the sum of an expression over a group of rows. SUM returns a DOUBLE PRECISION value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

The SUM aggregate function differs from the SUM analytic function, which computes the sum of an expression over a group of rows within a window.

**Behavior Type**

Immutable

**Syntax**

```sql
SUM ( [ ALL | DISTINCT ] expression )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Invokes the aggregate function for all rows in the group (default)</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Invokes the aggregate function for all distinct non-null values of the expression found in the group</td>
</tr>
<tr>
<td>expression</td>
<td>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.</td>
</tr>
</tbody>
</table>

**Overflow Handling**

If you encounter data overflow when using `SUM()`, use `SUM_FLOAT` which converts the data to a floating point. By default, Vertica allows silent numeric overflow when you call this function.
on numeric data types. For more information on this behavior and how to change it, see Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG.

Example

The following query returns the total sum of the product_cost column.

```sql
=> SELECT SUM(product_cost) AS cost FROM product_dimension;
   cost
  -------
  9042850
(1 row)
```

See Also

- AVG [Aggregate]
- COUNT [Aggregate]

SUM_FLOAT [Aggregate]

Computes the sum of an expression over a group of rows and returns a DOUBLE PRECISION value.

Behavior Type

Immutable

Syntax

```sql
SUM_FLOAT ( [ ALL | DISTINCT ] expression )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Invokes the aggregate function for all rows in the group (default).</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Invokes the aggregate function for all distinct non-null values of the expression found in the group.</td>
</tr>
</tbody>
</table>
Overflow Handling
By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG.

Example

The following query returns the floating-point sum of the average price from the product table:

```
=> SELECT SUM_FLOAT(average_competitor_price) AS cost FROM product_dimension;
         cost
    --------
     18181102  
(1 row)
```

VAR_POP [Aggregate]

Evaluates the population variance for each member of the group. This is defined as the sum of squares of the difference of \textit{expression} from the mean of \textit{expression}, divided by the number of remaining rows:

\[
\frac{(\text{SUM}(\text{expression} \times \text{expression}) - \text{SUM}(\text{expression}) \times \text{SUM}(\text{expression}) \div \text{COUNT}(\text{expression})) \div \text{COUNT}(\text{expression})}{\text{COUNT}(\text{expression})}
\]

Behavior Type

Immutable

Syntax

\texttt{VAR\_POP ( expression )}

Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. VAR\_POP returns the same |
data type as expression.

Related Functions

This aggregate function differs from analytic function VAR_POP, which computes the population variance of the current row with respect to the group of rows within a window.

Examples

The following example returns the population variance for each household ID in the customer table.

```
=> SELECT VAR_POP(household_id) FROM customer_dimension;

var_pop
-------------------------
74847050.8168393
(1 row)
```

VAR_SAMP [Aggregate]

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of expression from the mean of expression divided by the number of remaining rows minus 1:

\[
\frac{(\text{SUM}(\text{expression}^2) - \text{SUM}(\text{expression}) \ast \text{SUM}(\text{expression}) / \text{COUNT}(\text{expression})) / \text{COUNT}(\text{expression})}{\text{COUNT}(\text{expression}) - 1}
\]

Behavior Type

Immutable

Syntax

```
VAR_SAMP ( expression )
```

Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be |
implicitly converted to a numeric data type. VAR_SAMP returns the same data type as expression.

Related Functions

- VAR_SAMP is semantically identical to nonstandard function VARIANCE, which is provided for compatibility with other databases.
- This aggregate function differs from analytic function VAR_SAMP, which computes the sample variance of the current row with respect to the group of rows within a window.

Examples

The following example returns the sample variance for each household ID in the customer table.

```sql
=> SELECT VAR_SAMP(household_id) FROM customer_dimension;
  var_samp
------------------
  74848598.0106764  
(1 row)
```

See Also

VARIANCE [Aggregate]

VARIANCE [Aggregate]

Evaluates the sample variance for each row of the group. This is defined as the sum of squares of the difference of expression from the mean of expression divided by the number of remaining rows minus 1.

```
(SUM(expression*expression) - SUM(expression) * SUM(expression) / COUNT(expression)) / (COUNT(expression) - 1)
```

Behavior Type

Immutable
## Syntax

VARIANCE ( expression )

### Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. VARIANCE returns the same data type as expression. |

### Related Functions

The nonstandard function VARIANCE is provided for compatibility with other databases. It is semantically identical to VAR_SAMP.

This aggregate function differs from analytic function VARIANCE, which computes the sample variance of the current row with respect to the group of rows within a window.

### Examples

The following example returns the sample variance for each household ID in the customer table.

```sql
=> SELECT VARIANCE(household_id) FROM customer_dimension;

<table>
<thead>
<tr>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>74848598.0106764</td>
</tr>
</tbody>
</table>

(1 row)
```

### See Also

- [Analytic Functions](#)
- [VAR_SAMP [Aggregate]](#)
- [SQL Analytics](#)
Analytic Functions

Note: All analytic functions in this section with an aggregate counterpart are appended with [Analytics] in the heading to avoid confusion between the two function types.

Vertica analytics are SQL functions based on the ANSI 99 standard. These functions handle complex analysis and reporting tasks—for example:

- Rank the longest-standing customers in a particular state.
- Calculate the moving average of retail volume over a specified time.
- Find the highest score among all students in the same grade.
- Compare the current sales bonus that salespersons received against their previous bonus.

Analytic functions return aggregate results but they do not group the result set. They return the group value multiple times, once per record. You can sort group values, or partitions, using a window ORDER BY clause, but the order affects only the function result set, not the entire query result set.

Syntax

**General**

```
analytic-function ( arguments ) OVER(  
  ...[ window-partition-clause ]  
  ...[ window-order-clause [ window-frame-clause ] ]  
)
```

**With named window**

```
analytic-function ( arguments ) OVER(  
  ...[ named-window [ window-frame-clause ] ]  
)
```

Parameters

```
analytic-function (arguments)  
A Vertica analytic function and its arguments.
```
OVER | Specifies how to partition, sort, and window frame function input with respect to the current row. The input data is the result set that the query returns after it evaluates FROM, WHERE, GROUP BY, and HAVING clauses.

An empty OVER clause provides the best performance for single threaded queries on a single node.

**window-partition-clause** | Groups input rows according to one or more columns or expressions.

If you omit this clause, no grouping occurs and the analytic function processes all input rows as a single partition.

**window-order-clause** | Optionally specifies how to sort rows that are supplied to the analytic function. If the OVER clause also includes a partition clause, rows are sorted within each partition.

**window-frame-clause** | Only valid for some analytic functions, specifies as input a set of rows relative to the row that is currently being evaluated by the analytic function. After the function processes that row and its window, Vertica advances the current row and adjusts the window boundaries accordingly.

**named-window** | The name of a window that you define in the same query with a [window name clause](#). This definition encapsulates window partitioning and sorting. Named windows are useful when the query invokes multiple analytic functions with similar OVER clauses.

A window name clause cannot specify a window frame clause. However, you can qualify the named window in an OVER clause with a window frame clause.

### Requirements

The following requirements apply to analytic functions:
All require an OVER clause. Each function has its own OVER clause requirements. For example, you can supply an empty OVER clause for some analytic aggregate functions such as SUM. For other functions, window frame and order clauses might be required, or might be invalid.

- Analytic functions can be invoked only in a query's SELECT and ORDER BY clauses.
- Analytic functions cannot be nested. For example, the following query is not allowed:

```sql
=> SELECT MEDIAN(RANK() OVER(ORDER BY sal) OVER()).
```

- WHERE, GROUP BY and HAVING operators are technically not part of the analytic function. However, they determine input to that function.

**See Also**

- SQL Analytics
- GROUP BY Queries

**Window Partition Clause**

When specified, a window partition clause divides the rows of the function input based on user-provided expressions. If no expression is provided, the partition clause can improve query performance by using parallelism.

Window partitioning is similar to the GROUP BY clause except that it returns only one result row per input row. If you omit specifying a window partition clause, all input rows are treated as a single partition.

When used with analytic functions, results are computed per partition and start over again (reset) at the beginning of each subsequent partition.

**Syntax**

```sql
{ PARTITION BY expression[,...] | PARTITION BEST | PARTITION NODES }
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PARTITION BY</strong> (expression)</td>
<td>Expression on which to sort the partition, where (expression) can be a column, constant, or an arbitrary expression formed on columns. Use PARTITION BY for analytic functions with specific partitioning requirements.</td>
</tr>
<tr>
<td><strong>PARTITION BEST</strong></td>
<td>Use parallelism to improve performance for multi-threaded queries across multiple nodes. OVER(PARTITION BEST) provides the best performance on multi-threaded queries across multiple nodes. The following considerations apply to using PARTITION BEST:</td>
</tr>
<tr>
<td></td>
<td>- Use PARTITION BEST for analytic functions that have no partitioning requirements and are thread safe—for example, a one-to-many transform.</td>
</tr>
<tr>
<td></td>
<td>- Do not use PARTITION BEST on user-defined transform functions (UDTFs) that are not thread-safe. Doing so can produce an error or incorrect results. If a UDTF is not thread safe, use PARTITION NODES.</td>
</tr>
<tr>
<td><strong>PARTITION NODES</strong></td>
<td>Use parallelism to improve performance for single-threaded queries across multiple nodes. OVER(PARTITION NODES) provides the best performance on single-threaded queries across multiple nodes.</td>
</tr>
</tbody>
</table>

### Examples

See [Window Partitioning](#) in Analyzing Data.

### Window Order Cause

Specifies how to sort rows that are supplied to the analytic function. If the OVER clause also includes a window partition clause, rows are sorted within each partition.
The window order clause only specifies order within a window result set. The query can have its own ORDER BY clause outside the OVER clause. This has precedence over the window order clause and orders the final result set.

An window order clause also creates a default window frame if none is explicitly specified.

Syntax

```
ORDER BY { expression[ sort-qualifiers ] } [...]  
```

```
sort-qualifiers = 
[ ASC | DESC ]  
[ NULLS { FIRST | LAST | AUTO } ]
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>A column, constant, or arbitrary expression formed on columns, on which to sort input rows.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC</td>
<td>Specifies the ordering sequence as ascending (default) or descending.</td>
</tr>
<tr>
<td>DESC</td>
<td>Specifies whether to position null values first or last. Default positioning depends on whether the sort order is ascending or descending:</td>
</tr>
<tr>
<td>NULLS</td>
<td>Specifies whether to position null values first or last. Default positioning depends on whether the sort order is ascending or descending:</td>
</tr>
<tr>
<td>{FIRST</td>
<td>LAST</td>
</tr>
<tr>
<td></td>
<td>Descending default: NULLS FIRST</td>
</tr>
<tr>
<td>If you specify NULLS AUTO, Vertica chooses the positioning that is most efficient for this query, either either NULLS FIRST or NULLS LAST.</td>
<td></td>
</tr>
<tr>
<td>If you omit all sort qualifiers, Vertica uses ASC NULLS LAST.</td>
<td></td>
</tr>
<tr>
<td>For more information, see:</td>
<td></td>
</tr>
<tr>
<td>null sort order</td>
<td></td>
</tr>
<tr>
<td>Runtime Sorting of NULL Values in Analytic Functions</td>
<td></td>
</tr>
</tbody>
</table>
Examples

See Window Ordering in Analyzing Data.

Window Frame Clause

Specifies a window frame, which comprises a set of rows relative to the row that is currently being evaluated by the analytic function. After the function processes that row and its window, Vertica advances the current row and adjusts the window boundaries accordingly. If the OVER clause also specifies a partition, Vertica also checks that window boundaries do not cross partition boundaries. This process repeats until the function evaluates the last row of the last partition.

Syntax

\[
\{ \text{ROWS} \mid \text{RANGE} \} \\
\{ \text{BETWEEN start-point AND end-point} \} \mid \text{start-point} \\
\text{start-point \slash end-point} = \\
\{ \text{UNBOUNDED} \mid \text{PRECEDING} \mid \text{FOLLOWING} \} \\
\mid \text{CURRENT ROW} \\
\mid \text{constant-value} \{ \text{PRECEDING} \mid \text{FOLLOWING} \}
\]

Parameters

<table>
<thead>
<tr>
<th>ROWS</th>
<th>RANGE</th>
<th>Specifies whether Vertica determines window frame dimensions as physical or logical offsets from the current row. See ROWS versus RANGE below for details.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETWEEN start-point AND end-point</td>
<td>Specifies the window's first and last rows, where start-point and end-point can be one of the following (discussed in detail below):</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UNBOUNDED {PRECEDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CURRENT ROW</td>
</tr>
</tbody>
</table>
\begin{itemize}
\item \textit{constant-value} \{\texttt{PRECEDING} \mid \texttt{FOLLOWING}\}
\end{itemize}

\textit{start-point} must resolve to a row or value that is less than or equal to \textit{end-point}.

<table>
<thead>
<tr>
<th>Window Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{UNBOUNDED PRECEDING}</td>
<td>Specifies that the window frame extends to the current partition's first row.</td>
</tr>
<tr>
<td>\texttt{start-point}</td>
<td>If \texttt{ROWS} or \texttt{RANGE} specifies only a start point, Vertica uses the current row as the end point and creates the window frame accordingly. In this case, \textit{start-point} must resolve to a row that is less than or equal to the current row.</td>
</tr>
<tr>
<td>\texttt{UNBOUNDED FOLLOWING}</td>
<td>Specifies that the window frame extends to the current partition's last row.</td>
</tr>
<tr>
<td>\texttt{CURRENT ROW}</td>
<td>Specifies the current row or value as the window's start or end point.</td>
</tr>
<tr>
<td>\texttt{constant-value} {\texttt{PRECEDING} \mid \texttt{FOLLOWING}}</td>
<td>Specifies a constant value or expression that evaluates to a constant value. The value specifies a physical or logical offset from the current row, depending on whether you specify \texttt{ROWS} or \texttt{RANGE}. Other dependencies also pertain, depending whether you specify \texttt{ROWS} and \texttt{RANGE}. See \texttt{ROWS versus RANGE} below for details.</td>
</tr>
</tbody>
</table>

**Requirements**

In order to specify a window frame, the \texttt{OVER} must also specify a \texttt{window order} (\texttt{ORDER BY}) clause. If the \texttt{OVER} clause omits specifying a window frame, the function creates a default window that extends from the current row to the first row in the current partition. This is equivalent to the following clause:

\texttt{RANGE UNBOUNDED PRECEDING AND CURRENT ROW}

**ROWS versus RANGE**

The window frame's offset from the current row can be physical or logical:
• ROWS specifies the window's start-point and end-point as a number of rows relative to the current row. If start-point and end-point are expressed as constant values, the value must evaluate to a positive integer.

• RANGE specifies the window as a logical offset such as time. The range value must match the window order (ORDER BY) clause data type: NUMERIC, DATE/TIME, FLOAT or INTEGER.

Use of ROWS or RANGE imposes specific requirements on setting the window's start and end points as constant values:

Setting constant values for ROWS
The constant must evaluate to a positive INTEGER.

Setting constant values for RANGE
The following requirements apply:

• The constant must evaluate to a positive numeric value or INTERVAL literal.

• If the constant evaluates to a NUMERIC value, the ORDER BY column type must be a NUMERIC data type.

• If the constant evaluates to an INTERVAL DAY TO SECOND subtype, the ORDER BY column type must be one of the following: TIMESTAMP, TIME, DATE, or INTERVAL DAY TO SECOND.

• If the constant evaluates to an INTERVAL YEAR TO MONTH, the ORDER BY column type must be one of the following: TIMESTAMP, DATE, or INTERVAL YEAR TO MONTH.

• The window order clause can specify only one expression.

Examples

See Window Framing in Analyzing Data

Window Name Clause

Defines a named window that specifies window partition and order clauses for an analytic function. This window is specified in the function’s OVER clause. Named windows can be useful when you write queries that invoke multiple analytic functions with similar OVER clauses—for example, they use the same partition (PARTITION BY) clauses.
Syntax

```
WINDOW window-name AS ( window-partition-clause [window-order-clause] )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WINDOW window-name</code></td>
<td>Specifies the window name. All window names must be unique within the same query.</td>
</tr>
<tr>
<td><code>window-partition-clause [window-order-clause]</code></td>
<td>Clauses to invoke when an OVER clause references this window. If the window definition omits a window order clause, the OVER clause can specify its own order clause.</td>
</tr>
</tbody>
</table>

Requirements

- A `WINDOW` clause cannot include a `window frame clause`.
- Each `WINDOW` clause within the same query must have a unique name.
- A `WINDOW` clause can reference another window that is already named. For example, the following query names window `w1` before `w2`. Thus, the `WINDOW` clause that defines `w2` can reference `w1`:

```
=> SELECT RANK() OVER(w1 ORDER BY sal DESC), RANK() OVER w2
    FROM EMP WINDOW w1 AS (PARTITION BY deptno), w2 AS (w1 ORDER BY sal);
```

Examples

See Named Windows in Analyzing Data.

See Also

Analytic Functions
AVG [Analytic]

Computes an average of an expression in a group within a window. AVG returns the same data type as the expression's numeric data type.

The AVG analytic function differs from the AVG aggregate function, which computes the average of an expression over a group of rows.

Behavior Type

Immutable

Syntax

```
AVG ( expression ) OVER ( ...
[ window-partition-clause ]
[ window-order-clause ]
[ window-frame-clause ] )
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Any data that can be implicitly converted to a numeric data type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Overflow Handling

By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG.

Examples

The following query finds the sales for that calendar month and returns a running/cumulative average (sometimes called a moving average) using the default window of RANGE UNBOUNDED PRECEDING AND CURRENT ROW:
=> SELECT calendar_month_number_in_year Mo, SUM(product_price) Sales,
    AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year):INTEGER Average
FROM product_dimension pd, date_dimension dm, inventory_fact if
WHERE dm.date_key = if.date_key AND pd.product_key = if.product_key GROUP BY Mo;

Mo | Sales | Average
---|-------|--------
1  | 23869547 | 23869547
2  | 19684661 | 21737184
3  | 22877913 | 22117374
4  | 22901263 | 22313346
5  | 23670676 | 22584812
6  | 22507600 | 22571943
7  | 21514089 | 22420821
8  | 24860684 | 22725804
9  | 21687795 | 22610470
10 | 23648921 | 22714315
11 | 21115910 | 22569005
12 | 24708317 | 22747281
(12 rows)

To return a moving average that is not a running (cumulative) average, the window can specify
ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING:

=> SELECT calendar_month_number_in_year Mo, SUM(product_price) Sales,
    AVG(SUM(product_price)) OVER (ORDER BY calendar_month_number_in_year
    ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING):INTEGER Average
FROM product_dimension pd, date_dimension dm, inventory_fact if
WHERE dm.date_key = if.date_key AND pd.product_key = if.product_key GROUP BY Mo;

Mo | Sales | Average
---|-------|--------
1  | 23869547 | 22117374
2  | 19684661 | 22313346
3  | 22877913 | 22584812
4  | 22901263 | 22571943
5  | 23670676 | 22420821
6  | 22507600 | 22725804
7  | 21514089 | 22610470
8  | 24860684 | 22569005
9  | 21687795 | 22747281
10 | 23648921 | 22714315
11 | 21115910 | 22747281
12 | 24708317 | 22747281
(12 rows)

See Also

- COUNT [Analytic]
- SUM [Analytic]
- SQL Analytics
BOOL_AND [Analytic]

Returns the Boolean value of an expression within a window. If all input values are true, BOOL_AND returns \texttt{t}. Otherwise, it returns \texttt{f}.

Behavior Type

Immutable

Syntax

\texttt{BOOL\_AND} ( \texttt{expression} ) \texttt{OVER} ( \\
... [ \texttt{window-partition-clause} ] \\
... [ \texttt{window-order-clause} ] \\
... [ \texttt{window-frame-clause} ] )

Parameters

| expression | A Boolean Data Type or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value. |
| OVER() | See Analytic Functions. |

Examples

The following example illustrates how you can use the BOO\_AND, BOO\_OR, and BOO\_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

\begin{verbatim}
=> CREATE TABLE employee(emptype VARCHAR, yearspaid VARCHAR);
CREATE TABLE
\end{verbatim}

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

\begin{verbatim}
=> INSERT INTO employee
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2015'
\end{verbatim}
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

```sql
=> SELECT DISTINCT emptype,
       BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
       BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
       BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;
```

<table>
<thead>
<tr>
<th>emptype</th>
<th>paidtwicein2014</th>
<th>paidonceortwicein2014</th>
<th>paidjustoncein2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>contractor1</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor2</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor3</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor4</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor5</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

(5 rows)

See Also

- **BOOL_AND [Aggregate]**
- **BOOL_OR [Analytic]**
- **BOOL_XOR [Analytic]**
- **Boolean Data Type**
BOOL_OR [Analytic]

Returns the Boolean value of an expression within a window. If at least one input value is true, BOOL_OR returns t. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

BOOL_OR ( expression ) OVER ( ... [ window-partition-clause ] ... [ window-order-clause ] ... [ window-frame-clause ] )

Parameters

| expression | A Boolean Data Type or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value. |
| OVER()     | See Analytic Functions. |

Examples

The following example illustrates how you can use the BOOL_AND, BOOL_OR, and BOOL_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

```sql
=> CREATE TABLE employee(emptype VARCHAR, yearspaid VARCHAR);
CREATE TABLE
```

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

```sql
=> INSERT INTO employee
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2015'
```
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

```sql
=> SELECT DISTINCT emptype,
    BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
    BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
    BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;
```

<table>
<thead>
<tr>
<th>emptype</th>
<th>paidtwicein2014</th>
<th>paidonceortwicein2014</th>
<th>paidjustoncein2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>contractor1</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor2</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor3</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor4</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor5</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

See Also

- **BOOL_OR** [Aggregate]
- **BOOL_AND** [Analytic]
- **BOOL_XOR** [Analytic]
- **Boolean Data Type**
BOOL_XOR [Analytic]

Returns the Boolean value of an expression within a window. If only one input value is true, BOOL_XOR returns \( t \). Otherwise, it returns \( f \).

Behavior Type

Immutable

Syntax

\[
\text{BOOL}_XOR \left( \text{expression} \right) \over \text{(... \{ window-partition-clause \} ... \{ window-order-clause \} ... \{ window-frame-clause \})}
\]

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>A <strong>Boolean Data Type</strong> or any non-Boolean data type that can be implicitly converted to a Boolean data type. The function returns a Boolean value.</td>
</tr>
<tr>
<td>OVER()</td>
<td>See <a href="#">Analytic Functions</a>.</td>
</tr>
</tbody>
</table>

Examples

The following example illustrates how you can use the BOOL_AND, BOOL_OR, and BOOL_XOR analytic functions. The sample table, employee, includes a column for type of employee and years paid.

```sql
=> \text{CREATE TABLE employee(emptype VARCHAR, yearspaid VARCHAR);}\
CREATE TABLE
```

Insert sample data into the table to show years paid. In more than one case, an employee could be paid more than once within one year.

```sql
=> \text{INSERT INTO employee}\
\text{SELECT } \text{'contractor1', '2014'}\
\text{UNION ALL}\
\text{SELECT } \text{'contractor2', '2015'}
```
UNION ALL
SELECT 'contractor3', '2014'
UNION ALL
SELECT 'contractor1', '2014'
UNION ALL
SELECT 'contractor2', '2014'
UNION ALL
SELECT 'contractor3', '2015'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor4', '2014'
UNION ALL
SELECT 'contractor5', '2015'
UNION ALL
SELECT 'contractor5', '2016';

Query the table. The result shows employees that were paid twice in 2014 (BOOL_AND), once or twice in 2014 (BOOL_OR), and specifically not more than once in 2014 (BOOL_XOR).

=> SELECT DISTINCT emptype,
    BOOL_AND(yearspaid='2014') OVER (PARTITION BY emptype) AS paidtwicein2014,
    BOOL_OR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidonceortwicein2014,
    BOOL_XOR(yearspaid='2014') OVER (PARTITION BY emptype) AS paidjustoncein2014
FROM employee;

<table>
<thead>
<tr>
<th>emptype</th>
<th>paidtwicein2014</th>
<th>paidonceortwicein2014</th>
<th>paidjustoncein2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>contractor1</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor2</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor3</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>contractor4</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>contractor5</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

(5 rows)

See Also

- **BOOL_XOR [Aggregate]**
- **BOOL_AND [Analytic]**
- **BOOL_OR [Analytic]**
- **Boolean Data Type**
CONDITIONAL_CHANGE_EVENT [Analytic]

Assigns an event window number to each row, starting from 0, and increments by 1 when the result of evaluating the argument expression on the current row differs from that on the previous row.

Behavior Type

Immutable

Syntax

CONDITIONAL_CHANGE_EVENT ( expression ) OVER ( ... [ window-partition-clause ] ... window-order-clause )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>SQL scalar expression that is evaluated on an input record. The result of expression can be of any data type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Notes

The analytic window-order-clause is required but the window-partition-clause is optional.

Example

```sql
=> SELECT CONDITIONAL_CHANGE_EVENT(bid)
    OVER (PARTITION BY symbol ORDER BY ts) AS cce
FROM TickStore;
```

The system returns an error when no ORDER BY clause is present:
=> SELECT CONDITIONAL_CHANGE_EVENT(bid) OVER (PARTITION BY symbol) AS cce FROM TickStore;

ERROR: conditional_change_event must contain an ORDER BY clause within its analytic clause

For more examples, see Event-Based Windows in Analyzing Data.

See Also

- CONDITIONAL_TRUE_EVENT [Analytic]
- ROW_NUMBER [Analytic]
- Time Series Analytics
- Event-Based Windows

CONDITIONAL_TRUE_EVENT [Analytic]

Assigns an event window number to each row, starting from 0, and increments the number by 1 when the result of the boolean argument expression evaluates true. For example, given a sequence of values for column a, as follows:

( 1, 2, 3, 4, 5, 6 )

CONDITIONAL_TRUE_EVENT(a > 3) returns 0, 0, 0, 1, 2, 3.

Behavior Type:

Immutable

Syntax

CONDITIONAL_TRUE_EVENT ( boolean-expression ) OVER
... ( [ window-partition-clause ]
... window-order-clause )
Parameters

<table>
<thead>
<tr>
<th>boolean-expression</th>
<th>SQL scalar expression that is evaluated on an input record, type BOOLEAN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Notes

The analytic `window-order-clause` is required but the `window-partition-clause` is optional.

Example

```sql
> SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6)
     OVER(PARTITION BY bid ORDER BY ts) AS cte
FROM Tickstore;
```

The system returns an error if the ORDER BY clause is omitted:

```sql
> SELECT CONDITIONAL_TRUE_EVENT(bid > 10.6)
     OVER(PARTITION BY bid) AS cte
FROM Tickstore;
```

ERROR: conditional_true_event must contain an ORDER BY clause within its analytic clause

For more examples, see Event-Based Windows in Analyzing Data.

See Also

- `CONDITIONAL_CHANGE_EVENT [Analytic]`
- Time Series Analytics
- Event-Based Windows
COUNT [Analytic]

Counts occurrences within a group within a window. If you specify * or some non-null constant, COUNT() counts all rows.

Behavior Type

Immutable

Syntax

COUNT ( expression ) OVER ( ...
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ]
)

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Returns the number of rows in each group for which the expression is not null. Can be any expression resulting in BIGINT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Example

Using the schema defined in Window Framing in Analyzing Data, the following COUNT function omits window order and window frame clauses; otherwise Vertica would treat it as a window aggregate. Think of the window of reporting aggregates as UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING.

```sql
=> SELECT deptno, sal, empno, COUNT(sal)
    OVER (PARTITION BY deptno) AS count FROM emp;
```

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
Using ORDER BY sal creates a moving window query with default window: RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

=> SELECT deptno, sal, empno, COUNT(sal) OVER (PARTITION BY deptno ORDER BY sal) AS count
FROM emp;

deptno | sal | empno | count
-------|-----|-------|-----
10     | 101 | 1     | 1   
10     | 104 | 4     | 2   
20     | 100 | 11    | 1   
20     | 109 | 7     | 4   
20     | 109 | 6     | 4   
20     | 109 | 8     | 4   
20     | 110 | 10    | 6   
20     | 110 | 9     | 6   
30     | 102 | 2     | 1   
30     | 103 | 3     | 2   
30     | 105 | 5     | 3   

Using the VMart schema, the following query finds the number of employees who make less than or equivalent to the hourly rate of the current employee. The query returns a running/cumulative average (sometimes called a moving average) using the default window of RANGE UNBOUNDED PRECEDING AND CURRENT ROW:

=> SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)
OVER (ORDER BY hourly_rate) AS moving_count
FROM employee_dimension;

last_name | hourly_rate | moving_count
----------|------------|--------------
Gauthier   | 6          | 4            
Taylor     | 6          | 4            
Jefferson  | 6          | 4            
Nielson    | 6          | 4            
McNulty    | 6.01       | 11           
Robinson   | 6.01       | 11           
Dobisz     | 6.01       | 11           
Williams   | 6.01       | 11           
Kramer     | 6.01       | 11           
Miller     | 6.01       | 11           
Wilson     | 6.01       | 11           
Vogel      | 6.02       | 14           
Moore      | 6.02       | 14           
Vogel      | 6.02       | 14           
Carcetti   | 6.03       | 19           
...
To return a moving average that is not also a running (cumulative) average, the window should specify \texttt{ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING}:

\[
\Rightarrow \text{SELECT employee_last_name AS "last_name", hourly_rate, COUNT(*)}
\text{OVER (ORDER BY hourly_rate ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)}
\text{AS moving_count from employee_dimension;}
\]

See Also

- \texttt{COUNT [Aggregate]}
- \texttt{AVG [Analytic]}
- \texttt{SUM [Analytic]}
- SQL Analytics

\textbf{CUME\_DIST [Analytic]}

Calculates the cumulative distribution, or relative rank, of the current row with regard to other rows in the same partition within a window.

\texttt{CUME\_DIST()} returns a number greater than 0 and less than or equal to 1, where the number represents the relative position of the specified row within a group of \(n\) rows. For a row \(x\) (assuming \texttt{ASC} ordering), the \texttt{CUME\_DIST} of \(x\) is the number of rows with values lower than or equal to the value of \(x\), divided by the number of rows in the partition. For example, in a group of three rows, the cumulative distribution values returned would be 1/3, 2/3, and 3/3.

\textbf{Note:} Because the result for a given row depends on the number of rows preceding that row in the same partition, you should always specify a \texttt{window-order-clause} when you call this function.

\textbf{Behavior Type}

\texttt{Immutable}

\textbf{Syntax}

\texttt{CUME\_DIST ( ) OVER (}
\texttt{... [ window-partition-clause ]}
\texttt{... window-order-clause )}
Parameters

OVER( )  See Analytic Functions.

Examples

The following example returns the cumulative distribution of sales for different transaction types within each month of the first quarter.

```sql
=> SELECT calendar_month_name AS month, tender_type, SUM(sales_quantity),
   CUME_DIST() OVER (PARTITION BY calendar_month_name ORDER BY SUM(sales_quantity)) AS CUME_DIST
FROM store.store_sales_fact JOIN date_dimension
USING(date_key) WHERE calendar_month_name IN ('January','February','March')
   AND tender_type NOT LIKE 'Other'
GROUP BY calendar_month_name, tender_type;
```

<table>
<thead>
<tr>
<th>month</th>
<th>tender_type</th>
<th>SUM</th>
<th>CUME_DIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>March</td>
<td>Credit</td>
<td>469858</td>
<td>0.25</td>
</tr>
<tr>
<td>March</td>
<td>Cash</td>
<td>470449</td>
<td>0.5</td>
</tr>
<tr>
<td>March</td>
<td>Check</td>
<td>473033</td>
<td>0.75</td>
</tr>
<tr>
<td>March</td>
<td>Debit</td>
<td>475103</td>
<td>1</td>
</tr>
<tr>
<td>January</td>
<td>Cash</td>
<td>441730</td>
<td>0.25</td>
</tr>
<tr>
<td>January</td>
<td>Debit</td>
<td>443922</td>
<td>0.5</td>
</tr>
<tr>
<td>January</td>
<td>Check</td>
<td>446297</td>
<td>0.75</td>
</tr>
<tr>
<td>January</td>
<td>Credit</td>
<td>450994</td>
<td>1</td>
</tr>
<tr>
<td>February</td>
<td>Check</td>
<td>425665</td>
<td>0.25</td>
</tr>
<tr>
<td>February</td>
<td>Debit</td>
<td>426726</td>
<td>0.5</td>
</tr>
<tr>
<td>February</td>
<td>Credit</td>
<td>430010</td>
<td>0.75</td>
</tr>
<tr>
<td>February</td>
<td>Cash</td>
<td>430767</td>
<td>1</td>
</tr>
</tbody>
</table>

(12 rows)

See Also

- PERCENT_RANK [Analytic]
- PERCENTILE_DISC [Analytic]
- SQL Analytics
DENSE_RANK [Analytic]

Within each window partition, ranks all rows in the query results set according to the order specified by the window's ORDER BY clause. A DENSE_RANK function returns a sequence of ranking numbers without any gaps.

DENSE_RANK executes as follows:

1. Sorts partition rows as specified by the ORDER BY clause.

2. Compares the ORDER BY values of the preceding row and current row and ranks the current row as follows:
   - If ORDER BY values are the same, the current row gets the same ranking as the preceding row.
     
     **Note:** Null values are considered equal. For detailed information on how null values are sorted, see NULL Sort Order.

   - If the ORDER BY values are different, DENSE_RANK increments or decrements the current row's ranking by 1, depending whether sort order is ascending or descending.

DENSE_RANK always changes the ranking by 1, so no gaps appear in the ranking sequence. The largest rank value is the number of unique ORDER BY values returned by the query.

Behavior Type

Immutable

Syntax

DENSE_RANK() OVER(  
... [ window-partition-clause ]  
... window-order-clause )

Parameters

OVER() See Analytic Functions.

See Analytic Functions
Compared with RANK

**RANK** leaves gaps in the ranking sequence, while **DENSE_RANK** does not. The example below compares the behavior of the two functions.

### Example

The following query invokes **RANK** and **DENSE_RANK** to rank customers by annual income. The two functions return different rankings, as follows:

- If `annual_salary` contains duplicate values, **RANK()** inserts duplicate rankings and then skips one or more values—for example, from 4 to 6 and 7 to 9.
- In the parallel column **Dense Rank**, **DENSE_RANK**() also inserts duplicate rankings, but leaves no gaps in the rankings sequence:

```sql
=> SELECT employee_region region, employee_key, annual_salary,
   RANK() OVER (PARTITION BY employee_region ORDER BY annual_salary) Rank,
   DENSE_RANK() OVER (PARTITION BY employee_region ORDER BY annual_salary) "Dense Rank"
FROM employee_dimension;
```

<table>
<thead>
<tr>
<th>region</th>
<th>employee_key</th>
<th>annual_salary</th>
<th>Rank</th>
<th>Dense Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>West</td>
<td>5248</td>
<td>1200</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>West</td>
<td>6880</td>
<td>1204</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>West</td>
<td>5700</td>
<td>1214</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>West</td>
<td>9857</td>
<td>1218</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>West</td>
<td>6014</td>
<td>1218</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>West</td>
<td>9221</td>
<td>1220</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>West</td>
<td>7646</td>
<td>1222</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>West</td>
<td>6621</td>
<td>1222</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>West</td>
<td>6488</td>
<td>1224</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>West</td>
<td>7659</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>7432</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>9905</td>
<td>1226</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>9021</td>
<td>1228</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>West</td>
<td>56</td>
<td>963104</td>
<td>2794</td>
<td>2152</td>
</tr>
<tr>
<td>West</td>
<td>100</td>
<td>992363</td>
<td>2795</td>
<td>2153</td>
</tr>
<tr>
<td>East</td>
<td>8353</td>
<td>1200</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>East</td>
<td>9743</td>
<td>1202</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>East</td>
<td>9975</td>
<td>1202</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>East</td>
<td>9205</td>
<td>1204</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>East</td>
<td>8894</td>
<td>1206</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>East</td>
<td>7740</td>
<td>1206</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>East</td>
<td>7324</td>
<td>1208</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>East</td>
<td>6505</td>
<td>1208</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>East</td>
<td>5404</td>
<td>1208</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>East</td>
<td>5010</td>
<td>1208</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>East</td>
<td>9114</td>
<td>1212</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
See Also

SQL Analytics

EXPONENTIAL_MOVING_AVERAGE [Analytic]

Calculates the exponential moving average (EMA) of expression $E$ with smoothing factor $X$. An EMA differs from a simple moving average in that it provides a more stable picture of changes to data over time.

The EMA is calculated by adding the previous EMA value to the current data point scaled by the smoothing factor, as in the following formula:

$$EMA = EMA_0 + (X \times (E - EMA_0))$$

where:

- $E$ is the current data point
- $EMA_0$ is the previous row's EMA value.
- $X$ is the smoothing factor.

This function also works at the row level. For example, EMA assumes the data in a given column is sampled at uniform intervals. If the users' data points are sampled at non-uniform intervals, they should run the time series gap filling and interpolation (GFI) operations before EMA().

Behavior Type

Immutable

Syntax

```sql
EXPONENTIAL_MOVING_AVERAGE (E, X) OVER ( ...
... [ window-partition-clause ]
... window-order-clause )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>The value whose average is calculated over a set of rows. Can be INTEGER, FLOAT or NUMERIC type and must be a constant.</td>
</tr>
<tr>
<td>$X$</td>
<td>A positive FLOAT value between 0 and 1 that is used as the smoothing factor.</td>
</tr>
<tr>
<td>OVER ()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Examples

The following example uses time series gap filling and interpolation (GFI) first in a subquery, and then performs an EXPONENTIAL_MOVING_AVERAGE operation on the subquery result.

Create a simple four-column table:

```sql
=> CREATE TABLE ticker(
  time TIMESTAMP,
  symbol VARCHAR(8),
  bid1 FLOAT,
  bid2 FLOAT );
```

Insert some data, including nulls, so GFI can do its interpolation and gap filling:

```sql
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'ABC', 60.45, 60.44);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'ABC', 60.49, 65.12);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'ABC', 57.78, 59.25);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'ABC', null, 65.12);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'ABC', 67.88, null);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:00', 'XYZ', 47.55, 40.15);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:01', 'XYZ', 44.35, 46.78);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:02', 'XYZ', 71.56, 75.78);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:03', 'XYZ', 85.55, 70.21);
=> INSERT INTO ticker VALUES ('2009-07-12 03:00:04', 'XYZ', 45.55, 58.65);
=> COMMIT;
```

Note: During gap filling and interpolation, Vertica takes the closest non null value on either side of the time slice and uses that value. For example, if you use a linear interpolation scheme and you do not specify IGNORE NULLS, and your data has one real value and one null, the result is null. If the value on either side is null, the result is null. See When Time Series Data Contains Nulls in Analyzing Data for details.

Query the table that you just created to you can see the output:
The following query processes the first and last values that belong to each 2-second time slice in table trades' column a. The query then calculates the exponential moving average of expression fv and lv with a smoothing factor of 50%:

```sql
=> SELECT symbol, slice_time, fv, lv,
    EXPONENTIAL_MOVING_AVERAGE(fv, 0.5)
    OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_first,
    EXPONENTIAL_MOVING_AVERAGE(lv, 0.5)
    OVER (PARTITION BY symbol ORDER BY slice_time) AS ema_last
FROM (SELECT symbol, slice_time,
    TS_FIRST_VALUE(bid1 IGNORE NULLS) AS fv,
    TS_LAST_VALUE(bid2 IGNORE NULLS) AS lv
    FROM ticker TIMESERIES slice_time AS '2 seconds'
    OVER (PARTITION BY symbol ORDER BY time) ) AS sq;
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>slice_time</th>
<th>fv</th>
<th>lv</th>
<th>ema_first</th>
<th>ema_last</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>2009-07-12 03:00:00</td>
<td>60.45</td>
<td>65.12</td>
<td>60.45</td>
<td>65.12</td>
</tr>
<tr>
<td>ABC</td>
<td>2009-07-12 03:00:02</td>
<td>57.78</td>
<td>65.12</td>
<td>59.115</td>
<td>65.12</td>
</tr>
<tr>
<td>ABC</td>
<td>2009-07-12 03:00:04</td>
<td>67.88</td>
<td>65.12</td>
<td>63.4975</td>
<td>65.12</td>
</tr>
<tr>
<td>XYZ</td>
<td>2009-07-12 03:00:00</td>
<td>47.55</td>
<td>46.78</td>
<td>47.55</td>
<td>46.78</td>
</tr>
<tr>
<td>XYZ</td>
<td>2009-07-12 03:00:02</td>
<td>71.56</td>
<td>70.21</td>
<td>59.555</td>
<td>58.495</td>
</tr>
<tr>
<td>XYZ</td>
<td>2009-07-12 03:00:04</td>
<td>45.55</td>
<td>58.65</td>
<td>52.5525</td>
<td>58.5725</td>
</tr>
</tbody>
</table>

(6 rows)

See Also

- TIMESERIES Clause
- Time Series Analytics
- SQL Analytics
FIRST_VALUE [Analytic]

Lets you select the first value of a table or partition (determined by the window-order-clause) without having to use a self join. This function is useful when you want to use the first value as a baseline in calculations.

Use FIRST_VALUE() with the window-order-clause to produce deterministic results. If no window is specified for the current row, the default window is UNBOUNDED PRECEDING AND CURRENT ROW.

Behavior Type

Immutable

Syntax

FIRST_VALUE ( expression [ IGNORE NULLS ] ) OVER ( 
    ... [ window-partition-clause ] 
    ... [ window-order-clause ] 
    ... [ window-frame-clause ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Expression to evaluate—or example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORE NULLS</td>
<td>Specifies to return the first non-null value in the set, or NULL if all values are NULL. If you omit this option and the first value in the set is null, the function returns NULL.</td>
</tr>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Examples

The following query asks for the first value in the partitioned day of week, and illustrates the potential nondeterministic nature of FIRST_VALUE():
The first value returned is January 31, 2003; however, the next time the same query is run, the first value might be January 24 or January 3, or the 10th or 17th. This is because the analytic ORDER BY column day_of_week returns rows that contain ties (multiple Fridays). These repeated values make the ORDER BY evaluation result nondeterministic, because rows that contain ties can be ordered in any way, and any one of those rows qualifies as being the first value of day_of_week.

<table>
<thead>
<tr>
<th>calendar_year</th>
<th>date_key</th>
<th>day_of_week</th>
<th>full_date_description</th>
<th>first_value</th>
</tr>
</thead>
</table>

(31 rows)

Note: The day_of_week results are returned in alphabetical order because of lexical rules. The fact that each day does not appear ordered by the 7-day week cycle (for example, starting with Sunday followed by Monday, Tuesday, and so on) has no affect on results.
To return deterministic results, modify the query so that it performs its analytic ORDER BY operations on a unique field, such as date_key:

```sql
=> SELECT calendar_year, date_key, day_of_week, full_date_description,
    FIRST_VALUE(full_date_description) OVER
    (PARTITION BY calendar_month_number_in_year ORDER BY date_key) AS "first_value"
FROM date_dimension WHERE calendar_year=2003;
```

FIRST_VALUE() returns a first value of January 1 for the January partition and the first value of February 1 for the February partition. Also, the full_date_description column contains no ties:

<table>
<thead>
<tr>
<th>calendar_year</th>
<th>date_key</th>
<th>day_of_week</th>
<th>full_date_description</th>
<th>first_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>1</td>
<td>Wednesday</td>
<td>January 1, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>8</td>
<td>Wednesday</td>
<td>January 8, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>10</td>
<td>Friday</td>
<td>January 10, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>17</td>
<td>Friday</td>
<td>January 17, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>22</td>
<td>Wednesday</td>
<td>January 22, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>24</td>
<td>Friday</td>
<td>January 24, 2003</td>
<td>January 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>32</td>
<td>Saturday</td>
<td>February 1, 2003</td>
<td>February 1, 2003</td>
</tr>
<tr>
<td>2003</td>
<td>33</td>
<td>Sunday</td>
<td>February 2, 2003</td>
<td>February 1, 2003</td>
</tr>
</tbody>
</table>

... (365 rows)
See Also

- LAST_VALUE [Analytic]
- TIME_SLICE
- SQL Analytics

LAG [Analytic]

Returns the value of the input expression at the given offset before the current row within a window. This function lets you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.

For information on getting the rows that follow, see LEAD.

Behavior Type

Immutable

Syntax

LAG ( expression[, offset ][, default ] ) OVER ( ...
... [ window-partition-clause ]
... window-order-clause )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>The expression to evaluate—for example, a constant, column, non-analytic function, function expression, or expressions involving any of these.</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Indicates how great is the lag. The default value is 1 (the previous row). This parameter must evaluate to a constant positive integer.</td>
</tr>
<tr>
<td>default</td>
<td>The value returned if offset falls outside the bounds of the table or partition. This value must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first</td>
</tr>
</tbody>
</table>
Examples

This example sums the current balance by date in a table and also sums the previous balance from the last day. Given the inputs that follow, the data satisfies the following conditions:

- For each some_id, there is exactly 1 row for each date represented by month_date.
- For each some_id, the set of dates is consecutive; that is, if there is a row for February 24 and a row for February 26, there would also be a row for February 25.
- Each some_id has the same set of dates.

```
=> CREATE TABLE balances (  
    month_date DATE,  
    current_bal INT,  
    some_id INT);

=> INSERT INTO balances values ('2009-02-24', 10, 1);
=> INSERT INTO balances values ('2009-02-25', 10, 1);
=> INSERT INTO balances values ('2009-02-26', 10, 1);
=> INSERT INTO balances values ('2009-02-24', 20, 2);
=> INSERT INTO balances values ('2009-02-25', 20, 2);
=> INSERT INTO balances values ('2009-02-26', 20, 2);
=> INSERT INTO balances values ('2009-02-24', 30, 3);
=> INSERT INTO balances values ('2009-02-25', 30, 3);
=> INSERT INTO balances values ('2009-02-26', 30, 3);

Now run the LAG() function to sum the current balance for each date and sum the previous balance from the last day:

```
=> SELECT month_date,  
    SUM(current_bal) as current_bal_sum,  
    SUM(previous_bal) as previous_bal_sum FROM  
    (SELECT month_date, current_bal,  
      LAG(current_bal, 1, 0) OVER  
      (PARTITION BY some_id ORDER BY month_date)  
      AS previous_bal FROM balances) AS subQ  
GROUP BY month_date ORDER BY month_date;
```

<table>
<thead>
<tr>
<th>month_date</th>
<th>current_bal_sum</th>
<th>previous_bal_sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-24</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>2009-02-25</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>2009-02-26</td>
<td>60</td>
<td>50</td>
</tr>
</tbody>
</table>

(3 rows)

Using the same example data, the following query would not be allowed because LAG() is nested inside an aggregate function:
The following example uses the **VMart database**. LAG first returns the annual income from the previous row, and then it calculates the difference between the income in the current row from the income in the previous row:

```sql
=> SELECT month_date, 
    SUM(current_bal) as current_bal_sum, 
    SUM(LAG(current_bal, 1, 0) OVER 
        (PARTITION BY some_id ORDER BY month_date)) AS previous_bal_sum
FROM some_table GROUP BY month_date ORDER BY month_date;
```

The next example uses both LEA and LAG to return the third row after the salary in the current row and fifth salary before the salary in the current row:

```sql
=> SELECT hire_date, employee_key, employee_last_name, 
    LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired", 
    LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
FROM employee_dimension ORDER BY hire_date, employee_key;
```
See Also

- **LEAD [Analytic]**
- **SQL Analytics**

**LAST_VALUE [Analytic]**

Lets you select the last value of a table or partition (determined by the `window-order-clause`) without having to use a self join. `LAST_VALUE` takes the last record from the partition after the window order clause. The function then computes the expression against the last record, and returns the results. This function is useful when you want to use the last value as a baseline in calculations.

Use `LAST_VALUE()` with the `window-order-clause` to produce deterministic results. If no window is specified for the current row, the default window is `UNBOUNDED PRECEDING AND CURRENT ROW`.

Tip: Due to default window semantics, `LAST_VALUE` does not always return the last value of a partition. If you omit `Window Frame Clause` from the analytic clause, `LAST_VALUE` operates on this default window. Although results can seem non-intuitive by not returning the bottom of the current partition, it returns the bottom of the window, which continues to change along with the current input row being processed. If you want to return the last value of a partition, use `UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`. See examples below.

**Behavior Type**

Immutable

**Syntax**

```
LAST_VALUE ( expression [ IGNORE NULLS ] ) OVER ( ... [ window-partition-clause ]
```
Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Expression to evaluate—for example, a constant, column, nonanalytic function, function expression, or expressions involving any of these.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORE NULLS</td>
<td>Specifies to return the last non-null value in the set, or NULL if all values are NULL. If you omit this option and the last value in the set is null, the function returns NULL.</td>
</tr>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Example

Using the schema defined in Window Framing in Analyzing Data, the following query does not show the highest salary value by department; instead it shows the highest salary value by department by salary.

```sql
=> SELECT deptno, sal, empno, LAST_VALUE(sal)
     OVER (PARTITION BY deptno ORDER BY sal) AS lv
FROM emp;
```

<table>
<thead>
<tr>
<th>deptno</th>
<th>sal</th>
<th>empno</th>
<th>lv</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>11</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>7</td>
<td>109</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>6</td>
<td>109</td>
</tr>
<tr>
<td>20</td>
<td>109</td>
<td>8</td>
<td>109</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>20</td>
<td>110</td>
<td>9</td>
<td>110</td>
</tr>
<tr>
<td>30</td>
<td>102</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>30</td>
<td>103</td>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>30</td>
<td>105</td>
<td>5</td>
<td>105</td>
</tr>
</tbody>
</table>
```

If you include the window frame clause ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING, LAST_VALUE() returns the highest salary by department, an accurate representation of the information:

```sql
=> SELECT deptno, sal, empno, LAST_VALUE(sal)
     OVER (PARTITION BY deptno ORDER BY sal
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lv
FROM emp;
```
For more examples, see `FIRST_VALUE()`.

See Also

- `FIRST_VALUE [Analytic]`
- `TIME_SLICE`
- `SQL Analytics`

**LEAD [Analytic]**

Returns values from the row after the current row within a window, letting you access more than one row in a table at the same time. This is useful for comparing values when the relative positions of rows can be reliably known. It also lets you avoid the more costly self join, which enhances query processing speed.

**Behavior Type**

Immutable

**Syntax**

```
LEAD ( expression[, offset ] [, default ] ) OVER ( ... [ window-partition-clause ] ... window-order-clause )
```
Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>The expression to evaluate—for example, a constant, column, non-analytic function, function expression, or expressions involving any of these.</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Is an optional parameter that defaults to 1 (the next row). This parameter must evaluate to a constant positive integer.</td>
</tr>
<tr>
<td>default</td>
<td>The value returned if offset falls outside the bounds of the table or partition. This value must be a constant value or an expression that can be evaluated to a constant; its data type is coercible to that of the first argument.</td>
</tr>
<tr>
<td>OVER()</td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>

Examples

LEAD finds the hire date of the employee hired just after the current row:

```sql
=> SELECT employee_region, hire_date, employee_key, employee_last_name,
    LEAD(hire_date, 1) OVER (PARTITION BY employee_region ORDER BY hire_date) AS "next_hired"
FROM employee_dimension ORDER BY employee_region, hire_date, employee_key;
```

<table>
<thead>
<tr>
<th>employee_region</th>
<th>hire_date</th>
<th>employee_key</th>
<th>employee_last_name</th>
<th>next_hired</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>1956-04-08</td>
<td>9218</td>
<td>Harris</td>
<td>1957-02-06</td>
</tr>
<tr>
<td>East</td>
<td>1957-02-06</td>
<td>7799</td>
<td>Stein</td>
<td>1957-05-25</td>
</tr>
<tr>
<td>East</td>
<td>1957-05-25</td>
<td>3687</td>
<td>Farmer</td>
<td>1957-06-26</td>
</tr>
<tr>
<td>East</td>
<td>1957-06-26</td>
<td>9474</td>
<td>Bauer</td>
<td>1957-08-18</td>
</tr>
<tr>
<td>East</td>
<td>1957-08-18</td>
<td>570</td>
<td>Jefferson</td>
<td>1957-08-24</td>
</tr>
<tr>
<td>East</td>
<td>1957-08-24</td>
<td>4363</td>
<td>Wilson</td>
<td>1958-02-17</td>
</tr>
<tr>
<td>East</td>
<td>1958-02-17</td>
<td>6457</td>
<td>McCabe</td>
<td>1958-06-26</td>
</tr>
<tr>
<td>East</td>
<td>1958-06-26</td>
<td>6196</td>
<td>Li</td>
<td>1958-07-16</td>
</tr>
<tr>
<td>East</td>
<td>1958-07-16</td>
<td>7749</td>
<td>Harris</td>
<td>1958-09-18</td>
</tr>
<tr>
<td>East</td>
<td>1958-09-18</td>
<td>9678</td>
<td>Sanchez</td>
<td>1958-11-10</td>
</tr>
</tbody>
</table>

(10 rows)

The next example uses LEAD and LAG to return the third row after the salary in the current row and fifth salary before the salary in the current row.

```sql
=> SELECT hire_date, employee_key, employee_last_name,
    LEAD(hire_date, 1) OVER (ORDER BY hire_date) AS "next_hired",
    LAG(hire_date, 1) OVER (ORDER BY hire_date) AS "last_hired"
FROM employee_dimension ORDER BY hire_date, employee_key;
```

<table>
<thead>
<tr>
<th>hire_date</th>
<th>employee_key</th>
<th>employee_last_name</th>
<th>next_hired</th>
<th>last_hired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956-04-11</td>
<td>2694</td>
<td>Farmer</td>
<td>1956-05-12</td>
<td></td>
</tr>
<tr>
<td>1956-05-12</td>
<td>5486</td>
<td>Winkler</td>
<td>1956-09-18</td>
<td>1956-04-11</td>
</tr>
</tbody>
</table>
The following example returns employee name and salary, along with the next highest and lowest salaries.

```sql
=> SELECT employee_last_name, annual_salary,
    NVL(LEAD(annual_salary) OVER (ORDER BY annual_salary),
        MIN(annual_salary) OVER()) "Next Highest",
    NVL(LAG(annual_salary) OVER (ORDER BY annual_salary),
        MAX(annual_salary) OVER()) "Next Lowest"
FROM employee_dimension;
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>Next Highest</th>
<th>Next Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nielson</td>
<td>1200</td>
<td>1200</td>
<td>995533</td>
</tr>
<tr>
<td>Lewis</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>Harris</td>
<td>1200</td>
<td>1202</td>
<td>1200</td>
</tr>
<tr>
<td>Robinson</td>
<td>1202</td>
<td>1202</td>
<td>1202</td>
</tr>
<tr>
<td>Garnett</td>
<td>1202</td>
<td>1202</td>
<td>1202</td>
</tr>
<tr>
<td>Weaver</td>
<td>1202</td>
<td>1202</td>
<td>1202</td>
</tr>
<tr>
<td>Nielson</td>
<td>1202</td>
<td>1202</td>
<td>1202</td>
</tr>
<tr>
<td>McNulty</td>
<td>1202</td>
<td>1204</td>
<td>1202</td>
</tr>
<tr>
<td>Farmer</td>
<td>1204</td>
<td>1204</td>
<td>1202</td>
</tr>
<tr>
<td>Martin</td>
<td>1204</td>
<td>1204</td>
<td>1204</td>
</tr>
</tbody>
</table>

(10 rows)

The next example returns, for each assistant director in the employees table, the hire date of the director hired just after the director on the current row. For example, Jackson was hired on 2016-12-28, and the next director hired was Bauer:

```sql
=> SELECT employee_last_name, hire_date,
    LEAD(hire_date, 1) OVER (ORDER BY hire_date DESC) as "NextHired"
FROM employee_dimension WHERE job_title = 'Assistant Director';
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>hire_date</th>
<th>NextHired</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackson</td>
<td>2016-12-28</td>
<td>2016-12-26</td>
</tr>
<tr>
<td>Bauer</td>
<td>2016-12-26</td>
<td>2016-12-11</td>
</tr>
<tr>
<td>Miller</td>
<td>2016-12-11</td>
<td>2016-12-07</td>
</tr>
<tr>
<td>Fortin</td>
<td>2016-12-07</td>
<td>2016-11-27</td>
</tr>
<tr>
<td>Harris</td>
<td>2016-11-27</td>
<td>2016-11-15</td>
</tr>
<tr>
<td>Goldberg</td>
<td>2016-11-15</td>
<td></td>
</tr>
</tbody>
</table>

(5 rows)
See Also

- LAG [Analytic]
- SQL Analytics

MAX [Analytic]

Returns the maximum value of an expression within a window. The return value has the same type as the expression data type.

The analytic functions MIN() and MAX() can operate with Boolean values. The MAX() function acts upon a Boolean Data Type or a value that can be implicitly converted to a Boolean value. If at least one input value is true, MAX() returns $t$ (true). Otherwise, it returns $f$ (false). In the same scenario, the MIN() function returns $t$ (true) if all input values are true. Otherwise, it returns $f$.

Behavior Type

Immutable

Syntax

```
MAX ( expression ) OVER ( ...
... [ window-partition-clause ]
... [ window-order-clause ]
... [ window-frame-clause ]
)
```

Parameters

| expression | Any expression for which the maximum value is calculated, typically a column reference. |
| OVER()     | See Analytic Functions. |

Vertica Analytic Database (9.0.x)
Examples

The following query computes the deviation between the employees' annual salary and the maximum annual salary in Massachusetts:

```sql
=> SELECT employee_state, annual_salary,
    MAX(annual_salary)
    OVER(PARTITION BY employee_state ORDER BY employee_key) max,
    MAX(annual_salary) - MAX(annual_salary)
    OVER(PARTITION BY employee_state ORDER BY employee_key) diff
FROM employee_dimension
WHERE employee_state = 'MA';
```

<table>
<thead>
<tr>
<th>employee_state</th>
<th>annual_salary</th>
<th>max</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>1918</td>
<td>995533</td>
<td>-993615</td>
</tr>
<tr>
<td>MA</td>
<td>2058</td>
<td>995533</td>
<td>-993475</td>
</tr>
<tr>
<td>MA</td>
<td>2586</td>
<td>995533</td>
<td>-992947</td>
</tr>
<tr>
<td>MA</td>
<td>2500</td>
<td>995533</td>
<td>-993033</td>
</tr>
<tr>
<td>MA</td>
<td>1318</td>
<td>995533</td>
<td>-994215</td>
</tr>
<tr>
<td>MA</td>
<td>2072</td>
<td>995533</td>
<td>-993461</td>
</tr>
<tr>
<td>MA</td>
<td>2656</td>
<td>995533</td>
<td>-992877</td>
</tr>
<tr>
<td>MA</td>
<td>2148</td>
<td>995533</td>
<td>-993385</td>
</tr>
<tr>
<td>MA</td>
<td>2366</td>
<td>995533</td>
<td>-993167</td>
</tr>
<tr>
<td>MA</td>
<td>2664</td>
<td>995533</td>
<td>-992869</td>
</tr>
</tbody>
</table>

(10 rows)

The following example shows you the difference between the MIN and MAX analytic functions when you use them with a Boolean value. The sample creates a table with two columns, adds two rows of data, and shows sample output for MIN and MAX.

```sql
CREATE TABLE min_max_functions (emp VARCHAR, torf BOOL);

INSERT INTO min_max_functions VALUES ('emp1', 1);
INSERT INTO min_max_functions VALUES ('emp1', 0);

SELECT DISTINCT emp,
    min(torf) OVER (PARTITION BY emp) AS worksasbooleanand,
    Max(torf) OVER (PARTITION BY emp) AS worksasbooleanor
FROM min_max_functions;
```

<table>
<thead>
<tr>
<th>emp</th>
<th>worksasbooleanand</th>
<th>worksasbooleanor</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp1</td>
<td>f</td>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)
See Also

- SQL Analytics
- MAX [Aggregate]
- MIN [Analytic]

MEDIAN [Analytic]

For each row, returns the median value of a value set within each partition. MEDIAN determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type, and returns that data type.

MEDIAN is an alias of PERCENTILE_CONT [Analytic] with an argument of 0.5 (50%).

Behavior Type

Immutable

Syntax

MEDIAN ( expression ) OVER ( [ window-partition-clause ] )

Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the middle value or an interpolated value that would be the middle value once the values are sorted. Null values are ignored in the calculation. |
| OVER()     | If the OVER clause specifies window-partition-clause, MEDIAN groups input rows according to one or more columns or expressions. If this clause is omitted, no grouping occurs and MEDIAN processes all input rows as a single partition. |
Examples

See Calculating a Median Value

See Also

- PERCENTILE_CONT [Analytic]
- SQL Analytics

MIN [Analytic]

Returns the minimum value of an expression within a window. The return value has the same type as the expression data type.

The analytic functions MIN() and MAX() can operate with Boolean values. The MAX() function acts upon a Boolean Data Type or a value that can be implicitly converted to a Boolean value. If at least one input value is true, MAX() returns t (true). Otherwise, it returns f (false). In the same scenario, the MIN() function returns t (true) if all input values are true. Otherwise, it returns f.

Behavior Type

Immutable

Syntax

```
MAX ( expression ) OVER ( ...
   ... [ window-partition-clause ]
   ... [ window-order-clause ]
   ... [ window-frame-clause ]
)
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Any expression for which the minimum value is calculated, typically a column reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>
Examples

The following example shows how you can query to determine the deviation between the employees' annual salary and the minimum annual salary in Massachusetts:

```sql
=> SELECT employee_state, annual_salary, 
   MIN(annual_salary) 
   OVER(PARTITION BY employee_state ORDER BY employee_key) min, 
   annual_salary - MIN(annual_salary) 
   OVER(PARTITION BY employee_state ORDER BY employee_key) diff 
FROM employee_dimension 
WHERE employee_state = 'MA';
```

<table>
<thead>
<tr>
<th>employee_state</th>
<th>annual_salary</th>
<th>min</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>1918</td>
<td>1204</td>
<td>714</td>
</tr>
<tr>
<td>MA</td>
<td>2058</td>
<td>1204</td>
<td>854</td>
</tr>
<tr>
<td>MA</td>
<td>2586</td>
<td>1204</td>
<td>1382</td>
</tr>
<tr>
<td>MA</td>
<td>2500</td>
<td>1204</td>
<td>1296</td>
</tr>
<tr>
<td>MA</td>
<td>1318</td>
<td>1204</td>
<td>114</td>
</tr>
<tr>
<td>MA</td>
<td>2072</td>
<td>1204</td>
<td>868</td>
</tr>
<tr>
<td>MA</td>
<td>2656</td>
<td>1204</td>
<td>1452</td>
</tr>
<tr>
<td>MA</td>
<td>2148</td>
<td>1204</td>
<td>944</td>
</tr>
<tr>
<td>MA</td>
<td>2366</td>
<td>1204</td>
<td>1162</td>
</tr>
<tr>
<td>MA</td>
<td>2664</td>
<td>1204</td>
<td>1460</td>
</tr>
</tbody>
</table>

(10 rows)

The following example shows you the difference between the MIN and MAX analytic functions when you use them with a Boolean value. The sample creates a table with two columns, adds two rows of data, and shows sample output for MIN and MAX.

```sql
CREATE TABLE min_max_functions (emp VARCHAR, torf BOOL);
INSERT INTO min_max_functions VALUES ('emp1', 1);
INSERT INTO min_max_functions VALUES ('emp1', 0);

SELECT DISTINCT emp, 
   min(torf) OVER (PARTITION BY emp) AS worksasbooleanand, 
   Max(torf) OVER (PARTITION BY emp) AS worksasbooleanor 
FROM min_max_functions;
```

<table>
<thead>
<tr>
<th>emp</th>
<th>worksasbooleanand</th>
<th>worksasbooleanor</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp1</td>
<td>f</td>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)
See Also

- SQL Analytics
- MIN [Aggregate]
- MAX [Analytic]

NTILE [Analytic]

Equally divides an ordered data set (partition) into a \{value\} number of subsets within a window, where the subsets are numbered 1 through the value in parameter \textit{constant-value}. For example, if \textit{constant-value}=4 and the partition contains 20 rows, NTILE divides the partition rows into four equal subsets of five rows. NTILE assigns each row to a subset by giving row a number from 1 to 4. The rows in the first subset are assigned 1, the next five are assigned 2, and so on.

If the number of partition rows is not evenly divisible by the number of subsets, the rows are distributed so no subset is more than one row larger than any other subset, and the lowest subsets have extra rows. For example, if \textit{constant-value}=4 and the number of rows = 21, the first subset has six rows, the second subset has five rows, and so on.

If the number of subsets is greater than the number of rows, then a number of subsets equal to the number of rows is filled, and the remaining subsets are empty.

Behavior Type

Immutable

Syntax

```
NTILE ( constant-value ) OVER ( 
... [ window-partition-clause ]
... window-order-clause )
```

Parameters

<table>
<thead>
<tr>
<th>\textit{constant-value}</th>
<th>Specifies the number of subsets, where \textit{constant-value} must</th>
</tr>
</thead>
</table>
resolve to a positive constant for each partition.

OVER()  See Analytic Functions.

Examples

The following query assigns each month's sales total into one of four subsets:

```sql
=> SELECT calendar_month_name AS MONTH,
    SUM(sales_quantity),
    NTILE(4) OVER (ORDER BY SUM(sales_quantity)) AS NTILE
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
GROUP BY calendar_month_name
ORDER BY NTILE;
```

<table>
<thead>
<tr>
<th>MONTH</th>
<th>SUM</th>
<th>NTILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>November</td>
<td>2040726</td>
<td>1</td>
</tr>
<tr>
<td>June</td>
<td>2088528</td>
<td>1</td>
</tr>
<tr>
<td>February</td>
<td>2134708</td>
<td>1</td>
</tr>
<tr>
<td>April</td>
<td>2181767</td>
<td>2</td>
</tr>
<tr>
<td>January</td>
<td>2229220</td>
<td>2</td>
</tr>
<tr>
<td>October</td>
<td>2316363</td>
<td>2</td>
</tr>
<tr>
<td>September</td>
<td>2323914</td>
<td>3</td>
</tr>
<tr>
<td>March</td>
<td>2354409</td>
<td>3</td>
</tr>
<tr>
<td>August</td>
<td>2387017</td>
<td>3</td>
</tr>
<tr>
<td>July</td>
<td>2417239</td>
<td>4</td>
</tr>
<tr>
<td>May</td>
<td>2492182</td>
<td>4</td>
</tr>
<tr>
<td>December</td>
<td>2531842</td>
<td>4</td>
</tr>
</tbody>
</table>

(12 rows)

See Also

- PERCENTILE_CONT [Analytic]
- WIDTH_BUCKET
- SQL Analytics

NTH_VALUE [Analytic]

Returns the value evaluated at the row that is the \( n \)th row of the window (counting from 1). If the specified row does not exist, NTH_VALUE returns NULL.
Behavior Type

Immutable

Syntax

\[
\text{NTH\_VALUE}(\text{expression}, \text{row-number} \ [\text{IGNORE NULLS} \ ] ) \ \text{OVER}( \ 
\quad \cdots \ [\text{window-frame-clause} \ ] \\
\quad \cdots \ [\text{Window Order Cause} \ ] \ \\
\quad \cdots \ [\text{window-frame-clause} \ ]
\)
\]

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Expression to evaluate. The expression can be a constant, column name, nonanalytic function, function expression, or expressions that include any of these.</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-number</td>
<td>Specifies the row to evaluate, where \text{row-number} evaluates to an integer ( \geq 1 ).</td>
</tr>
<tr>
<td>IGNORE NULLS</td>
<td>Specifies to return the first non-NULL value in the set, or NULL if all values are NULL.</td>
</tr>
<tr>
<td>OVER()</td>
<td>See Analytic Functions.</td>
</tr>
</tbody>
</table>

Example

In the following example, for each tuple (current row) in table t1, the window frame clause defines the window as follows:

\[
\text{ORDER BY b ROWS BETWEEN 3 PRECEDING AND CURRENT ROW}
\]

For each window, \( n \) for \( n \)th value is \( a+1 \). \( a \) is the value of column \( a \) in the tuple.

\( \text{NTH\_VALUE} \) returns the result of the expression \( b+1 \), where \( b \) is the value of column \( b \) in the \( n \)th row, which is the \( a+1 \) row within the window.

\[
\Rightarrow \text{SELECT * FROM t1 ORDER BY a;}
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
</tr>
</tbody>
</table>

(7 rows)

```sql
=> SELECT NTH_VALUE(b+1, a+1) OVER (ORDER BY b ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) FROM t1;
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

(7 rows)

### PERCENT_RANK [Analytic]

Calculates the relative rank of a row for a given row in a group within a window by dividing that row’s rank less 1 by the number of rows in the partition, also less 1. PERCENT_RANK always returns values from 0 to 1 inclusive. The first row in any set has a PERCENT_RANK of 0. The return value is NUMBER.

\[
\frac{(\text{rank} - 1)}{([\text{rows}] - 1)}
\]

In the preceding formula, `rank` is the rank position of a row in the group and `rows` is the total number of rows in the partition defined by the `OVER()` clause.

### Behavior Type

Immutable

### Syntax

```
PERCENT_RANK ( ) OVER (  
  ... [ window-partition-clause ]  
  ... window-order-clause  )
```
Parameters

**OVER()**  
See Analytic Functions

Examples

The following example finds the percent rank of gross profit for different states within each month of the first quarter:

```
=> SELECT calendar_month_name AS MONTH, store_state,
     SUM(gross_profit_dollar_amount),
     PERCENT_RANK() OVER (PARTITION BY calendar_month_name
     ORDER BY SUM(gross_profit_dollar_amount)) AS PERCENT_RANK
FROM store.store_sales_fact JOIN date_dimension
USING(date_key)
JOIN store.store_dimension
USING (store_key)
WHERE calendar_month_name IN ('January','February','March')
AND store_state IN ('OR','IA','DC','NV','WI')
GROUP BY calendar_month_name, store_state
ORDER BY calendar_month_name, PERCENT_RANK;
```

<table>
<thead>
<tr>
<th>MONTH</th>
<th>store_state</th>
<th>SUM</th>
<th>PERCENT_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>February</td>
<td>IA</td>
<td>418490</td>
<td>0</td>
</tr>
<tr>
<td>February</td>
<td>OR</td>
<td>460588</td>
<td>0.25</td>
</tr>
<tr>
<td>February</td>
<td>DC</td>
<td>616553</td>
<td>0.5</td>
</tr>
<tr>
<td>February</td>
<td>WI</td>
<td>619204</td>
<td>0.75</td>
</tr>
<tr>
<td>February</td>
<td>NV</td>
<td>838039</td>
<td>1</td>
</tr>
<tr>
<td>January</td>
<td>OR</td>
<td>446528</td>
<td>0</td>
</tr>
<tr>
<td>January</td>
<td>IA</td>
<td>474501</td>
<td>0.25</td>
</tr>
<tr>
<td>January</td>
<td>DC</td>
<td>628496</td>
<td>0.5</td>
</tr>
<tr>
<td>January</td>
<td>WI</td>
<td>679382</td>
<td>0.75</td>
</tr>
<tr>
<td>January</td>
<td>NV</td>
<td>871824</td>
<td>1</td>
</tr>
<tr>
<td>March</td>
<td>IA</td>
<td>460282</td>
<td>0</td>
</tr>
<tr>
<td>March</td>
<td>OR</td>
<td>481935</td>
<td>0.25</td>
</tr>
<tr>
<td>March</td>
<td>DC</td>
<td>716663</td>
<td>0.5</td>
</tr>
<tr>
<td>March</td>
<td>WI</td>
<td>771575</td>
<td>0.75</td>
</tr>
<tr>
<td>March</td>
<td>NV</td>
<td>970878</td>
<td>1</td>
</tr>
</tbody>
</table>

(15 rows)

The following example calculates, for each employee, the percent rank of the employee's salary by their job title:

```
=> SELECT job_title, employee_last_name, annual_salary,
     PERCENT_RANK() OVER (PARTITION BY job_title ORDER BY annual_salary DESC) AS percent_rank
FROM employee_dimension
ORDER BY percent_rank, annual_salary;
```

<table>
<thead>
<tr>
<th>job_title</th>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>percent_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(15 rows)
## See Also

- `CUME_DIST [Analytic]`
- `SQL Analytics`

## PERCENTILE_CONT [Analytic]

An inverse distribution function where, for each row, `PERCENTILE_CONT` returns the value that would fall into the specified percentile among a set of values in each partition within a window. For example, if the argument to the function is 0.5, the result of the function is the median of the data set (50th percentile). `PERCENTILE_CONT` assumes a continuous distribution data model. NULL values are ignored.

`PERCENTILE_CONT` computes the percentile by first computing the row number where the percentile row would exist. For example:
If \( \text{row-number} \) is a whole number (within an error of 0.00001), the percentile is the value of row \( \text{row-number} \).

Otherwise, Vertica interpolates the percentile value between the value of the \( \text{CEILING} (\text{row-number}) \) row and the value of the \( \text{FLOOR} (\text{row-number}) \) row. In other words, the percentile is calculated as follows:

\[
( \text{CEILING} (\text{row-number}) - \text{row-number} ) \times ( \text{Value of FLOOR(\text{row-number}) row} ) \\
+ ( \text{row-number} - \text{FLOOR(\text{row-number})} ) \times ( \text{Value of CEILING(\text{row-number}) row} )
\]

Note: If the percentile value is 0.5, \text{PERCENTILE\_CONT} returns the same result set as the function \text{MEDIAN}.

Behavior Type

Immutable

Syntax

\text{PERCENTILE\_CONT ( percentile ) WITHIN GROUP ( ORDER BY expression [ ASC | DESC ] ) OVER ( ... [ window-partition-clause ] )}

Parameters

<table>
<thead>
<tr>
<th>\text{percentile}</th>
<th>Percentile value, a FLOAT constant that ranges from 0 to 1 (inclusive).</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{WITHIN GROUP (ORDER BY expression)}</td>
<td>Specifies how to sort data within each group. \text{ORDER BY} takes only one column/expression that must be \text{INTEGER}, \text{FLOAT}, \text{INTERVAL}, or \text{NUMERIC} data type. \text{NULL} values are discarded. &lt;br&gt;The \text{WITHIN GROUP(ORDER BY)} clause does not guarantee the order of the SQL result. To order the final result, use the SQL \text{ORDER BY} clause set.</td>
</tr>
<tr>
<td>\text{ASC</td>
<td>DESC}</td>
</tr>
<tr>
<td>\text{OVER()}</td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>
Examples

This query computes the median annual income per group for the first 300 customers in Wisconsin and the District of Columbia.

```sql
=> SELECT customer_state, customer_key, annual_income, 
   PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY annual_income) 
   OVER (PARTITION BY customer_state) AS PERCENTILE_CONT 
FROM customer_dimension 
WHERE customer_state IN ('DC', 'WI') 
AND customer_key < 300 
ORDER BY customer_state, customer_key;
```

<table>
<thead>
<tr>
<th>customer_state</th>
<th>customer_key</th>
<th>annual_income</th>
<th>PERCENTILE_CONT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>52</td>
<td>168312</td>
<td>483266.5</td>
</tr>
<tr>
<td>DC</td>
<td>118</td>
<td>798221</td>
<td>483266.5</td>
</tr>
<tr>
<td>WI</td>
<td>62</td>
<td>283043</td>
<td>377691</td>
</tr>
<tr>
<td>WI</td>
<td>139</td>
<td>472339</td>
<td>377691</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)

This query computes the median annual income per group for all customers in Wisconsin and the District of Columbia.

```sql
=> SELECT customer_state, customer_key, annual_income, 
   PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY annual_income) 
   OVER (PARTITION BY customer_state) AS PERCENTILE_CONT 
FROM customer_dimension 
WHERE customer_state IN ('DC', 'WI') 
ORDER BY customer_state, customer_key;
```

<table>
<thead>
<tr>
<th>customer_state</th>
<th>customer_key</th>
<th>annual_income</th>
<th>PERCENTILE_CONT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>52</td>
<td>168312</td>
<td>483266.5</td>
</tr>
<tr>
<td>DC</td>
<td>118</td>
<td>798221</td>
<td>483266.5</td>
</tr>
<tr>
<td>DC</td>
<td>622</td>
<td>220782</td>
<td>555088</td>
</tr>
<tr>
<td>DC</td>
<td>951</td>
<td>178453</td>
<td>555088</td>
</tr>
<tr>
<td>DC</td>
<td>972</td>
<td>961582</td>
<td>555088</td>
</tr>
<tr>
<td>DC</td>
<td>1286</td>
<td>760445</td>
<td>555088</td>
</tr>
<tr>
<td>DC</td>
<td>1434</td>
<td>44836</td>
<td>555088</td>
</tr>
<tr>
<td>WI</td>
<td>62</td>
<td>283043</td>
<td>377691</td>
</tr>
<tr>
<td>WI</td>
<td>139</td>
<td>472339</td>
<td>377691</td>
</tr>
<tr>
<td>WI</td>
<td>359</td>
<td>42242</td>
<td>517717</td>
</tr>
<tr>
<td>WI</td>
<td>364</td>
<td>867543</td>
<td>517717</td>
</tr>
<tr>
<td>WI</td>
<td>403</td>
<td>509031</td>
<td>517717</td>
</tr>
<tr>
<td>WI</td>
<td>455</td>
<td>32000</td>
<td>517717</td>
</tr>
<tr>
<td>WI</td>
<td>485</td>
<td>373129</td>
<td>517717</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1353 rows)
PERCENTILE_DISC [Analytic]

An inverse distribution function where, for each row, PERCENTILE_DISC returns the value that would fall into the specified percentile among a set of values in each partition within a window. PERCENTILE_DISC() assumes a discrete distribution data model. NULL values are ignored.

PERCENTILE_DISC examines the cumulative distribution values in each group until it finds one that is greater than or equal to the specified percentile. Vertica computes the percentile where, for each row, PERCENTILE_DISC outputs the first value of the WITHIN GROUP (ORDER BY) column whose CUME_DIST (cumulative distribution) value is >= the argument FLOAT value—for example, 0.4:

```
PERCENTILE_DISC(0.4) WITHIN GROUP (ORDER BY salary) OVER(PARTITION BY deptno)...
```

Given the following query:

```
SELECT CUME_DIST() OVER(ORDER BY salary) FROM table-name;
```

The smallest CUME_DIST value that is greater than 0.4 is also the PERCENTILE_DISC.

Behavior Type

Immutable

Syntax

```
PERCENTILE_DISC ( percentile ) WITHIN GROUP ( 
... ORDER BY expression [ ASC | DESC ] ) OVER ( 
... [ window-partition-clause ] )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>percentile</strong></td>
<td>Percentile value, a FLOAT constant that ranges from 0 to 1 (inclusive).</td>
</tr>
<tr>
<td><strong>WITHIN GROUP</strong></td>
<td>Specifies how to sort data within each group. ORDER BY takes only one</td>
</tr>
<tr>
<td><strong>(ORDER BY expression)</strong></td>
<td>column/expression that must be INTEGER, FLOAT, INTERVAL, or NUMERIC data</td>
</tr>
<tr>
<td></td>
<td>type. NULL values are discarded.</td>
</tr>
<tr>
<td></td>
<td>The WITHIN GROUP(ORDER BY) clause does not guarantee the order of the</td>
</tr>
<tr>
<td></td>
<td>SQL result. To order the final result, use the SQL ORDER BY clause set.</td>
</tr>
<tr>
<td>**ASC</td>
<td>DESC**</td>
</tr>
<tr>
<td><strong>OVER()</strong></td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>

Example

This query computes the 20th percentile annual income by group for first 300 customers in Wisconsin and the District of Columbia.

```
=> SELECT customer_state, customer_key, annual_income, 
   PERCENTILE_DISC(.2) WITHIN GROUP(ORDER BY annual_income) 
   OVER (PARTITION BY customer_state) AS PERCENTILE_DISC 
FROM customer_dimension 
WHERE customer_state IN ('DC','WI') 
AND customer_key < 300 
ORDER BY customer_state, customer_key;
```

<table>
<thead>
<tr>
<th>customer_state</th>
<th>customer_key</th>
<th>annual_income</th>
<th>PERCENTILE_DISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>104</td>
<td>658383</td>
<td>417092</td>
</tr>
<tr>
<td>DC</td>
<td>168</td>
<td>417092</td>
<td>417092</td>
</tr>
<tr>
<td>DC</td>
<td>245</td>
<td>670205</td>
<td>417092</td>
</tr>
<tr>
<td>WI</td>
<td>106</td>
<td>227279</td>
<td>227279</td>
</tr>
<tr>
<td>WI</td>
<td>127</td>
<td>703889</td>
<td>227279</td>
</tr>
<tr>
<td>WI</td>
<td>209</td>
<td>458607</td>
<td>227279</td>
</tr>
</tbody>
</table>

(6 rows)

See Also

- **CUME_DIST [Analytic]**
- **PERCENTILE_CONT [Analytic]**
**SQL Analytics**

**RANK [Analytic]**

Within each window partition, ranks all rows in the query results set according to the order specified by the window's ORDER BY clause.

RANK executes as follows:

1. Sorts partition rows as specified by the ORDER BY clause.

2. Compares the ORDER BY values of the preceding row and current row and ranks the current row as follows:
   - If ORDER BY values are the same, the current row gets the same ranking as the preceding row.
     
     **Note:** Null values are considered equal. For detailed information on how null values are sorted, see [NULL Sort Order](#).

   - If the ORDER BY values are different, DENSE_RANK increments or decrements the current row's ranking by 1, plus the number of consecutive duplicate values in the rows that precede it.

The largest rank value is the equal to the total number of rows returned by the query.

**Behavior Type**

Immutable

**Syntax**

\[
\text{RANK}() \text{ OVER(}
\text{ ... [ window-partition-clause ] }
\text{ ... window-order-clause )}
\]

**Parameters**

| OVER() | See Analytic Functions |
Compared with DENSE_RANK

RANK can leave gaps in the ranking sequence, while DENSE_RANK does not. For more information, see DENSE_RANK.

Examples

The following query ranks by state all company customers that have been customers since 2007. In rows where the customer_since dates are the same, RANK assigns the rows equal ranking. When the customer_since date changes, RANK skips one or more rankings—for example, within CA, from 12 to 14, and from 17 to 19.

```sql
=> SELECT customer_state, customer_name, customer_since,
    RANK() OVER (PARTITION BY customer_state ORDER BY customer_since) AS rank
FROM customer_dimension WHERE customer_type='Company' AND customer_since > '01/01/2007'
ORDER BY customer_state;

+----------+----------+------------+-----+
<table>
<thead>
<tr>
<th>customer_state</th>
<th>customer_name</th>
<th>customer_since</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ</td>
<td>Foodshop</td>
<td>2007-01-20</td>
<td>1</td>
</tr>
<tr>
<td>AZ</td>
<td>Goldstar</td>
<td>2007-08-11</td>
<td>2</td>
</tr>
<tr>
<td>CA</td>
<td>Metahope</td>
<td>2007-01-05</td>
<td>1</td>
</tr>
<tr>
<td>CA</td>
<td>Foodgen</td>
<td>2007-02-05</td>
<td>2</td>
</tr>
<tr>
<td>CA</td>
<td>Infohope</td>
<td>2007-02-09</td>
<td>3</td>
</tr>
<tr>
<td>CA</td>
<td>Foodcom</td>
<td>2007-02-19</td>
<td>4</td>
</tr>
<tr>
<td>CA</td>
<td>Amerihope</td>
<td>2007-02-22</td>
<td>5</td>
</tr>
<tr>
<td>CA</td>
<td>Infostar</td>
<td>2007-03-05</td>
<td>6</td>
</tr>
<tr>
<td>CA</td>
<td>Intracare</td>
<td>2007-03-14</td>
<td>7</td>
</tr>
<tr>
<td>CA</td>
<td>Infocare</td>
<td>2007-04-07</td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CO</td>
<td>Goldtech</td>
<td>2007-02-19</td>
<td>1</td>
</tr>
<tr>
<td>CT</td>
<td>Foodmedia</td>
<td>2007-02-11</td>
<td>1</td>
</tr>
<tr>
<td>CT</td>
<td>Metatech</td>
<td>2007-02-20</td>
<td>2</td>
</tr>
<tr>
<td>CT</td>
<td>Infocorp</td>
<td>2007-04-10</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

See Also

SQL Analytics

ROW_NUMBER [Analytic]

Assigns a sequence of unique numbers, starting from 1, to each row in a window partition. Use the optional window partition clause to group data into partitions before operating on it. For example:
SUM OVER (PARTITION BY col1, col2, ...)

Notes:

- **ROW_NUMBER()** is a Vertica extension, not part of the SQL-99 standard.
- **ROW_NUMBER** and **RANK** are generally interchangeable. **ROW_NUMBER** differs from **RANK** in that it assigns a unique ordinal number to each row in the ordered set, starting with 1.

Behavior Type

Immutable

Syntax

```sql
ROW_NUMBER() OVER (
    ... [ window-partition-clause ]
    ... window-order-clause )
```

Parameters

```sql
OVER()  See Analytic Functions
```

Examples

The following query partitions customers in the VMart table `customer_dimension` by occupation. It then ranks those customers according to the ordered set specified by the window partition clause.

```sql
=> SELECT occupation, customer_key, customer_since, annual_income,
       ROW_NUMBER() OVER (PARTITION BY occupation) AS customer_since_row_num
FROM public.customer_dimension
ORDER BY occupation, customer_since_row_num;
```

<table>
<thead>
<tr>
<th>occupation</th>
<th>customer_key</th>
<th>customer_since</th>
<th>annual_income</th>
<th>customer_since_row_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accountant</td>
<td>49985</td>
<td>1987-04-05</td>
<td>998685</td>
<td>1</td>
</tr>
<tr>
<td>Accountant</td>
<td>49977</td>
<td>1989-06-14</td>
<td>616235</td>
<td>2</td>
</tr>
<tr>
<td>Accountant</td>
<td>49929</td>
<td>1978-10-08</td>
<td>298965</td>
<td>3</td>
</tr>
<tr>
<td>Accountant</td>
<td>49913</td>
<td>1988-10-18</td>
<td>141013</td>
<td>4</td>
</tr>
</tbody>
</table>
| Accountant | 49844 | 1989-10-04 | 967475 | 5
| Accountant | 49839 | 1971-11-08 | 942459 | 6
| Accountant | 49823 | 1979-11-28 | 435959 | 7
| Accountant | 49817 | 1987-06-15 | 538732 | 8
| Accountant | 49733 | 1972-04-21 | 651928 | 9
| Accountant | 49707 | 1980-02-17 | 420219 | 10
| Acrobat     | 49912 | 2007-06-08 | 722953 | 1
| Acrobat     | 49908 | 1996-03-05 | 380288 | 2
| Acrobat     | 49833 | 2003-11-15 | 317918 | 3
| Acrobat     | 49770 | 1984-11-18 | 986536 | 4
| Acrobat     | 49725 | 1973-04-09 | 911864 | 5
| Acrobat     | 49641 | 1970-11-24 | 870287 | 6
| Acrobat     | 49685 | 1972-01-08 | 322062 | 7
| Acrobat     | 49577 | 1980-01-15 | 121727 | 8
| Acrobat     | 49575 | 1975-11-03 | 835388 | 9
| Actor       | 49974 | 2003-06-12 | 885346 | 1
| Actor       | 49937 | 1986-07-25 | 692557 | 2
| Actor       | 49889 | 1985-09-09 | 766587 | 3
| Actor       | 49850 | 1980-01-01 | 328270 | 4
| Actor       | 49820 | 1984-11-17 | 826861 | 5
| Actor       | 49780 | 1987-06-01 | 54853  | 6
| Actor       | 49760 | 2000-07-05 | 255977 | 7
| Actor       | 49698 | 1971-05-18 | 543584 | 8
| Actor       | 49676 | 1997-07-23 | 710498 | 9
| Actor       | 49631 | 1985-11-12 | 67353  | 10

See Also

- RANK [Analytic]
- SQL Analytics

STDDEV [Analytic]

Computes the statistical sample standard deviation of the current row with respect to the group within a window. STDDEV_SAMP returns the same value as the square root of the variance defined for the VAR_SAMP function:

\[
\text{STDDEV( expression )} = \sqrt{\text{VAR}_\text{SAMP}( \text{expression} )}
\]

When VAR_SAMP returns NULL, this function returns NULL.

Note: The nonstandard function STDDEV is provided for compatibility with other databases. It is semantically identical to STDDEV_SAMP.
Behavior Type

Immutable

Syntax

```
STDDEV ( expression ) OVER ( ...
  [ window-partition-clause ]
  [ window-order-clause ]
  [ window-frame-clause ] )
```

Parameters

| expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument. |
| OVER() | See Analytic Functions |

Example

The following example returns the standard deviations of salaries in the employee dimension table by job title Assistant Director:

```
=> SELECT employee_last_name, annual_salary,
    STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev"
FROM employee_dimension
WHERE job_title = 'Assistant Director';
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bauer</td>
<td>85003</td>
<td>NaN</td>
</tr>
<tr>
<td>Reyes</td>
<td>91051</td>
<td>4276.58181261624</td>
</tr>
<tr>
<td>Overstreet</td>
<td>53296</td>
<td>28278.6923394976</td>
</tr>
<tr>
<td>Gauthier</td>
<td>97216</td>
<td>19543.7184537642</td>
</tr>
<tr>
<td>Jones</td>
<td>82320</td>
<td>16928.0764028285</td>
</tr>
<tr>
<td>Fortin</td>
<td>56166</td>
<td>18400.2738421652</td>
</tr>
<tr>
<td>Carcetti</td>
<td>71135</td>
<td>16968.9453554483</td>
</tr>
<tr>
<td>Weaver</td>
<td>74419</td>
<td>15729.0799901852</td>
</tr>
<tr>
<td>Stein</td>
<td>85689</td>
<td>15040.590495309</td>
</tr>
<tr>
<td>McNulty</td>
<td>69423</td>
<td>14401.1524291943</td>
</tr>
<tr>
<td>Webber</td>
<td>99091</td>
<td>15256.3160166536</td>
</tr>
<tr>
<td>Meyer</td>
<td>74774</td>
<td>14588.6126417355</td>
</tr>
<tr>
<td>Garnett</td>
<td>82169</td>
<td>14008.7223268494</td>
</tr>
<tr>
<td>Roy</td>
<td>76974</td>
<td>13466.1270356647</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### STDDEV_POP [Analytic]

Computes the statistical population standard deviation and returns the square root of the population variance within a window. The `STDDEV_POP()` return value is the same as the square root of the `VAR_POP()` function:

\[
\text{STDDEV}_{\text{POP}}( \text{expression} ) = \sqrt{\text{VAR}_{\text{POP}}( \text{expression} )}
\]

When `VAR_POP` returns null, `STDDEV_POP` returns null.

### Behavior Type

Immutable

### Syntax

\[
\text{STDDEV}_{\text{POP}}( \text{expression} ) \over \text{window-partition-clause} \over \text{window-order-clause} \over \text{window-frame-clause}
\]

### Parameters

| `expression` | Any NUMERIC data type or any non-numeric data type that can be |
implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.

**OVER()**  See Analytic Functions.

### Examples

The following example returns the population standard deviations of salaries in the employee dimension table by job title Assistant Director:

```sql
=> SELECT employee_last_name, annual_salary, 
   STDDEV_POP(annual_salary) OVER (ORDER BY hire_date) as "stddev_pop"
   FROM employee_dimension WHERE job_title = 'Assistant Director';
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>stddev_pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goldberg</td>
<td>61859</td>
<td>0</td>
</tr>
<tr>
<td>Miller</td>
<td>79582</td>
<td>8861.5</td>
</tr>
<tr>
<td>Goldberg</td>
<td>74236</td>
<td>7422.74712548456</td>
</tr>
<tr>
<td>Campbell</td>
<td>66426</td>
<td>6850.22125099891</td>
</tr>
<tr>
<td>Moore</td>
<td>66630</td>
<td>6322.08223926257</td>
</tr>
<tr>
<td>Nguyen</td>
<td>53530</td>
<td>8356.55400006099</td>
</tr>
<tr>
<td>Harris</td>
<td>74115</td>
<td>8122.72288970008</td>
</tr>
<tr>
<td>Lang</td>
<td>59981</td>
<td>8053.54776538731</td>
</tr>
<tr>
<td>Farmer</td>
<td>60597</td>
<td>7858.70140687825</td>
</tr>
<tr>
<td>Nguyen</td>
<td>78941</td>
<td>8360.63150784682</td>
</tr>
</tbody>
</table>
```

### See Also

- STDDEV_POP [Aggregate]
- SQL Analytics

### STDDEV_SAMP [Analytic]

Computes the statistical sample standard deviation of the current row with respect to the group within a window. STDDEV_SAMP's return value is the same as the square root of the variance defined for the VAR_SAMP function:

\[
\text{STDDEV( expression )} = \sqrt{\text{VAR_SAMP( expression )}}
\]

When VAR_SAMP returns NULL, STDDEV_SAMP returns NULL.

**Note:** STDDEV_SAMP() is semantically identical to the nonstandard function, STDDEV().
Behavior Type

Immutable

Syntax

$\text{STDDEV}_\text{SAMP} \ ( \text{expression} \ ) \ \text{OVER} \ (\ldots \ [\ \text{window-partition-clause} \ ] \ldots \ [\ \text{window-order-clause} \ ] \ldots \ [\ \text{window-frame-clause} \ ])$

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>

Examples

The following example returns the sample standard deviations of salaries in the employee dimension table by job title Assistant Director:

```sql
=> SELECT employee_last_name, annual_salary, 
   STDDEV(annual_salary) OVER (ORDER BY hire_date) as "stddev_samp"
FROM employee_dimension WHERE job_title = 'Assistant Director';
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>annual_salary</th>
<th>stddev_samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bauer</td>
<td>85003</td>
<td>NaN</td>
</tr>
<tr>
<td>Reyes</td>
<td>91051</td>
<td>4276.58181261624</td>
</tr>
<tr>
<td>Overstreet</td>
<td>53296</td>
<td>20278.6923394976</td>
</tr>
<tr>
<td>Gauthier</td>
<td>97216</td>
<td>19543.7184537642</td>
</tr>
<tr>
<td>Jones</td>
<td>82320</td>
<td>16928.0764028285</td>
</tr>
<tr>
<td>Fortin</td>
<td>56166</td>
<td>18400.2738421652</td>
</tr>
<tr>
<td>Carcetti</td>
<td>71135</td>
<td>16968.9453554483</td>
</tr>
<tr>
<td>Weaver</td>
<td>74419</td>
<td>15729.0709901852</td>
</tr>
<tr>
<td>Stein</td>
<td>85689</td>
<td>15040.590495309</td>
</tr>
<tr>
<td>McNulty</td>
<td>69423</td>
<td>14401.1524291943</td>
</tr>
<tr>
<td>McNulty</td>
<td>99091</td>
<td>15256.3160166536</td>
</tr>
<tr>
<td>Meyer</td>
<td>74774</td>
<td>14588.6126417355</td>
</tr>
<tr>
<td>Garnett</td>
<td>82169</td>
<td>14000.7223268494</td>
</tr>
<tr>
<td>Roy</td>
<td>76974</td>
<td>13466.1270356647</td>
</tr>
<tr>
<td>Dobisz</td>
<td>83486</td>
<td>13040.4887828347</td>
</tr>
</tbody>
</table>
```
See Also

- Analytic Functions
- STDDEV [Analytic]
- STDDEV [Aggregate]
- STDDEV_SAMP [Aggregate]
- SQL Analytics

SUM [Analytic]

Computes the sum of an expression over a group of rows within a window. It returns a DOUBLE PRECISION value for a floating-point expression. Otherwise, the return value is the same as the expression data type.

Behavior Type

Immutable

Syntax

```
SUM ( expression ) OVER (
  ...
  [ window-partition-clause ]
  ...
  [ window-order-clause ]
  ...
  [ window-frame-clause ]
)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression</code></td>
<td>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.</td>
</tr>
<tr>
<td><code>OVER()</code></td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>
Overflow Handling

If you encounter data overflow when using SUM, use `SUM_FLOAT` which converts data to a floating point. By default, Vertica allows silent numeric overflow when you call this function on numeric data types. For more information on this behavior and how to change it, see Numeric Data Type Overflow with SUM, SUM_FLOAT, and AVG.

Examples

The following query returns the cumulative sum all of the returns made to stores in January:

```sql
=> SELECT calendar_month_name AS month, transaction_type, sales_quantity, 
   SUM(sales_quantity) OVER (PARTITION BY calendar_month_name ORDER BY date_dimension.date_key) AS SUM 
   FROM store.store_sales_fact JOIN date_dimension 
   USING(date_key) WHERE calendar_month_name IN ('January') 
   AND transaction_type= 'return';
```

<table>
<thead>
<tr>
<th>month</th>
<th>transaction_type</th>
<th>sales_quantity</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>3</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>3</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>6</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>6</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>3</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>6</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>9</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>6</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>8</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>2</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>4</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>5</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>7</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>8</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>4</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>10</td>
<td>651</td>
</tr>
<tr>
<td>January</td>
<td>return</td>
<td>6</td>
<td>651</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See Also

- **SUM [Aggregate]**
- **Numeric Data Types**
- **SQL Analytics**

**VAR_POP [Analytic]**

Returns the statistical population variance of a non-null set of numbers (nulls are ignored) in a group within a window. Results are calculated by the sum of squares of the difference of `expression` from the mean of `expression`, divided by the number of rows remaining:

\[
\frac{\left(\text{SUM}( \text{expression} \times \text{expression} \right) - \text{SUM}( \text{expression} \times \text{expression} ) / \text{COUNT}( \text{expression} )}{\text{COUNT}( \text{expression} )}
\]

**Behavior Type**

Immutable

**Syntax**

```
VAR_POP ( expression ) OVER ( ...
...[ window-partition-clause ]
...[ window-order-clause ]
...[ window-frame-clause ] )
```

**Parameters**

<table>
<thead>
<tr>
<th><code>expression</code></th>
<th>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OVER()</code></td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>
Examples

The following example calculates the cumulative population in the store orders fact table of sales in January 2007:

```sql
=> SELECT date_ordered,
       VAR_POP(SUM(total_order_cost)) OVER (ORDER BY date_ordered) "var_pop"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-01-01' AND '2007-01-31'
GROUP BY s.date_ordered;
```

<table>
<thead>
<tr>
<th>date_ordered</th>
<th>var_pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-01-01</td>
<td>0</td>
</tr>
<tr>
<td>2007-01-02</td>
<td>89870400</td>
</tr>
<tr>
<td>2007-01-03</td>
<td>3470302472</td>
</tr>
<tr>
<td>2007-01-04</td>
<td>4466755450.6875</td>
</tr>
<tr>
<td>2007-01-05</td>
<td>3816904780.80078</td>
</tr>
<tr>
<td>2007-01-06</td>
<td>25438212385.25</td>
</tr>
<tr>
<td>2007-01-07</td>
<td>22168747513.1016</td>
</tr>
<tr>
<td>2007-01-08</td>
<td>23445191812.7344</td>
</tr>
<tr>
<td>2007-01-09</td>
<td>39292879603.1113</td>
</tr>
<tr>
<td>2007-01-10</td>
<td>48080574326.9609</td>
</tr>
</tbody>
</table>

(10 rows)

See Also

- **VAR_POP [Aggregate]**
- **SQL Analytics**

VAR_SAMP [Analytic]

Returns the sample variance of a non-NULL set of numbers (NULL values in the set are ignored) for each row of the group within a window. Results are calculated as follows:

```sql
(SUM( expression * expression ) - SUM( expression ) * SUM( expression ) / COUNT( expression ) )
/ (COUNT( expression ) - 1 )
```

This function and **VARIANCE** differ in one way: given an input set of one element, VARIANCE returns 0 and VAR_SAMP returns NULL.

Behavior Type

Immutable
Syntax

```sql
VAR_SAMP ( expression ) OVER ( ...
    [ window-partition-clause ]
    [ window-order-clause ]
    [ window-frame-clause ]
)
```  

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER()</td>
<td>See Analytic Functions</td>
</tr>
</tbody>
</table>

Null Handling

- `VAR_SAMP` returns the sample variance of a set of numbers after it discards the NULL values in the set.

- If the function is applied to an empty set, then it returns NULL.

Examples

The following example calculates the sample variance in the store orders fact table of sales in December 2007:

```sql
=> SELECT date_ordered,
    VAR_SAMP(SUM(total_order_cost))
    OVER (ORDER BY date_ordered) "var_samp"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY s.date_ordered;
```

<table>
<thead>
<tr>
<th>date_ordered</th>
<th>var_samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-12-01</td>
<td>NaN</td>
</tr>
<tr>
<td>2007-12-02</td>
<td>90642601088</td>
</tr>
<tr>
<td>2007-12-03</td>
<td>48830548449.3359</td>
</tr>
<tr>
<td>2007-12-04</td>
<td>32740062504.2461</td>
</tr>
<tr>
<td>2007-12-05</td>
<td>3210319112.6992</td>
</tr>
<tr>
<td>2007-12-06</td>
<td>26274166814.668</td>
</tr>
<tr>
<td>2007-12-07</td>
<td>23017490251.9062</td>
</tr>
<tr>
<td>2007-12-08</td>
<td>21099374085.1406</td>
</tr>
</tbody>
</table>
See Also

- VARIANCE [Analytic]
- VAR_SAMP [Aggregate]
- SQL Analytics

VARIANCE [Analytic]

Returns the sample variance of a non-NULL set of numbers (NULL values in the set are ignored) for each row of the group within a window. Results are calculated as follows:

\[
\frac{\left( \sum (expression \times expression) - \sum(expression) \times \sum(expression) \right)}{(\text{COUNT(expression)} - 1)}\]

VARIANCE returns the variance of \( expression \), which is calculated as follows:

- 0 if the number of rows in \( expression \) = 1
- VAR_SAMP if the number of rows in \( expression \) > 1

Note: The nonstandard function VARIANCE is provided for compatibility with other databases. It is semantically identical to VAR_SAMP.

Behavior Type

Immutable

Syntax

\[\text{VAR_SAMP}(expression) \text{ OVER (}} \ldots [ \text{window-partition-clause}] \ldots [ \text{window-order-clause}] \ldots [ \text{window-frame-clause}] \)
Parameters

expression | Any NUMERIC data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.

OVER() | See Analytic Functions

Examples

The following example calculates the cumulative variance in the store orders fact table of sales in December 2007:

```sql
SELECT date_ordered,
       VARIANCE(SUM(total_order_cost)) OVER (ORDER BY date_ordered) "variance"
FROM store.store_orders_fact s
WHERE date_ordered BETWEEN '2007-12-01' AND '2007-12-31'
GROUP BY s.date_ordered;
```

<table>
<thead>
<tr>
<th>date_ordered</th>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-12-01</td>
<td>NaN</td>
</tr>
<tr>
<td>2007-12-02</td>
<td>2259129762</td>
</tr>
<tr>
<td>2007-12-03</td>
<td>1809012182.33301</td>
</tr>
<tr>
<td>2007-12-04</td>
<td>35138165568.25</td>
</tr>
<tr>
<td>2007-12-05</td>
<td>26644110029.3003</td>
</tr>
<tr>
<td>2007-12-06</td>
<td>25943125234</td>
</tr>
<tr>
<td>2007-12-07</td>
<td>23178202223.9648</td>
</tr>
<tr>
<td>2007-12-08</td>
<td>21940268901.1431</td>
</tr>
<tr>
<td>2007-12-09</td>
<td>21487676799.6108</td>
</tr>
<tr>
<td>2007-12-10</td>
<td>21521358853.4331</td>
</tr>
</tbody>
</table>

(10 rows)

See Also

- VAR_SAMP [Analytic]
- VARIANCE [Aggregate]
- VAR_SAMP [Aggregate]
- SQL Analytics
Current Load Source

The current load source function returns the source file name.

CURRENT_LOAD_SOURCE

Returns the file name used during the COPY statement.

Behavior Type
Stable

Syntax

CURRENT_LOAD_SOURCE()

Behavior

- If the function is called outside of the context of a COPY statement, it returns NULL.
- If the function is called by a UDL that does not set the source, it returns the string <unknown>.
- This function is not supported for COPY LOCAL.

Examples

This example creates a table and populates column c3 with the names of the two separate files being loaded.

```sql
=> CREATE TABLE t (c1 integer, c2 varchar(50), c3 varchar(200));
CREATE TABLE

=> COPY t (c1, c2, c3 AS CURRENT_LOAD_SOURCE()) FROM '/home/load_file_1' ON exampledb_node02,
  '/home/load_file_2' ON exampledb_node03 DELIMITER ',';
```
See Also

- copy
**Date/Time Functions**

Date and time functions perform conversion, extraction, or manipulation operations on date and time data types and can return date and time information.

**Usage**

Functions that take TIME or TIMESTAMP inputs come in two variants:

- TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE
- TIME WITHOUT TIME ZONE or TIMESTAMP WITHOUT TIME ZONE

For brevity, these variants are not shown separately.

The + and * operators come in commutative pairs; for example, both DATE + INTEGER and INTEGER + DATE. We show only one of each such pair.

**Daylight Savings Time Considerations**

When adding an INTERVAL value to (or subtracting an INTERVAL value from) a TIMESTAMP WITH TIME ZONE value, the days component advances (or decrements) the date of the TIMESTAMP WITH TIME ZONE by the indicated number of days. Across daylight saving time changes (with the session time zone set to a time zone that recognizes DST), this means INTERVAL '1 day' does not necessarily equal INTERVAL '24 hours'.

For example, with the session time zone set to CST7CDT:

```plaintext
TIMESTAMP WITH TIME ZONE '2014-04-02 12:00-07' + INTERVAL '1 day'
```

produces

```plaintext
TIMESTAMP WITH TIME ZONE '2014-04-03 12:00-06'
```

Adding INTERVAL '24 hours' to the same initial TIMESTAMP WITH TIME ZONE produces

```plaintext
TIMESTAMP WITH TIME ZONE '2014-04-03 13:00-06',
```

This result occurs because there is a change in daylight saving time at 2014-04-03 02:00 in time zone CST7CDT.
Date/Time Functions in Transactions

Certain date/time functions such as CURRENT_TIMESTAMP and NOW return the start time of the current transaction; for the duration of that transaction, they return the same value. Other date/time functions such as TIMEOFDAY always return the current time.

See Also

Template Patterns for Date/Time Formatting

ADD_MONTHS

Adds the specified number of months to a date and returns the sum as a DATE. The function returns the last day of the month if one of the following conditions is true:

- The start-date argument specifies the last day of the month. For example, 2015-01-28 +12 returns 2016-01-29.
- The start-date argument's day component is greater than the last day of the month returned by ADD_MONTHS. For example, 2015-01-29 +1 returns 2015-02-28.

Otherwise, ADD_MONTHS returns a date with the same day component as start-date. Thus, 2016-03-15 +2 returns 2016-05-15.

Behavior Type

- Immutable if the start-date argument is a TIMESTAMP or DATE
- Stable if the start-date argument is a TIMESTAMPTZ

Syntax

ADD_MONTHS ( start-date, num-months );
Parameters

<table>
<thead>
<tr>
<th>start-date</th>
<th>The date to process, an expression that evaluates to one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

| num-months | An integer expression that specifies the number of months to add to or subtract from start-date. |

Examples

Add one month to the current date:

```sql
=> SELECT CURRENT_DATE Today;
   Today
   ------------
   2016-05-05
   (1 row)
```

`VMart=>` SELECT ADD_MONTHS(CURRENT_TIMESTAMP,1);

```sql
ADD_MONTHS
   ------------
   2016-06-05
   (1 row)
```

Subtract four months from the current date:

```sql
=> SELECT ADD_MONTHS(CURRENT_TIMESTAMP, -4);
   ADD_MONTHS
   ------------
   2016-01-05
   (1 row)
```

Add one month to January 31 2016:

```sql
=> SELECT ADD_MONTHS('31-Jan-2016':TIMESTAMP, 1) "Leap Month";
   Leap Month
   ------------
   2016-02-29
   (1 row)
```
The following example sets the timezone to EST; it then adds 24 months to a TIMESTAMPTZ that specifies a PST time zone, so ADD_MONTHS takes into account the time change:

```sql
=> SET TIME ZONE 'America/New_York';
SET
VMart=> SELECT ADD_MONTHS('2008-02-29 23:30 PST':TIMESTAMPTZ, 24);
ADD_MONTHS
--------
2010-03-01
(1 row)
```

**AGE_IN_MONTHS**

Returns the difference in months between two dates, expressed as an integer.

**Behavior Type**

- Immutable if both date arguments are of data type TIMESTAMP
- Stable if either date is a TIMESTAMPTZ or only one argument is supplied

**Syntax**

```sql
AGE_IN_MONTHS ( [ date1,] date2 )
```

**Parameters**

<table>
<thead>
<tr>
<th><code>date1</code></th>
<th><code>date2</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the boundaries of the period to measure. If you supply only one argument, Vertica sets <code>date1</code> to <code>CURRENT_DATE</code>. Both parameters must evaluate to one of the following data types:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

If `date1 < date2`, AGES_IN_MONTHS returns a negative value.
Examples

Get the age in months of someone born March 2 1972, as of June 21 1990:

```sql
=> SELECT AGE_IN_MONTHS('1990-06-21'::TIMESTAMP, '1972-03-02'::TIMESTAMP);
  AGE_IN_MONTHS
  --------
   219
(1 row)
```

Get the age in months of someone who was born November 21 1939, as of today:

```sql
=> SELECT AGE_IN_MONTHS ('1939-11-21'::DATE);
  AGE_IN_MONTHS
  --------
   930
(1 row)
```

AGE_IN_YEARS

Returns the difference in years between two dates, expressed as an integer.

Behavior Type

- Immutable if both date arguments are of data type TIMESTAMP
- Stable if either date is a TIMESTAMPTZ or only one argument is supplied

Syntax

```
AGE_IN_YEARS( [ date1, ] date2 )
```

Parameters

<table>
<thead>
<tr>
<th>date1</th>
<th>date2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the boundaries of the period to measure. If you supply only one argument, Vertica sets date1 to CURRENT_DATE. Both parameters must evaluate to one of the following data types:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DATE</td>
</tr>
</tbody>
</table>
- TIMESTAMP
- TIMESTAMPTZ

If `date1 < date2`, `AGES_IN_YEARS` returns a negative value.

**Examples**

Get the age of someone born March 2 1972, as of June 21 1990:

```sql
=> SELECT AGE_IN_YEARS('1990-06-21':TIMESTAMP, '1972-03-02':TIMESTAMP);

AGE_IN_YEARS
---------
   18
(1 row)
```

Get the age of someone who was born November 21 1939, as of today:

```sql
=> SELECT AGE_IN_YEARS('1939-11-21':DATE);

AGE_IN_YEARS
---------
   77
(1 row)
```

**CLOCK_TIMESTAMP**

Returns a value of type `TIMESTAMP` WITH TIMEZONE that represents the current system-clock time.

`CLOCK_TIMESTAMP` uses the date and time supplied by the operating system on the server to which you are connected, which should be the same across all servers. The value changes each time you call it.

**Behavior Type**

Volatile

**Syntax**

`CLOCK_TIMESTAMP()`
Examples

The following command returns the current time on your system:

```
SELECT CLOCK_TIMESTAMP() "Current Time";
```

```
Current Time
2010-09-23 11:41:23.33772-04
(1 row)
```

Each time you call the function, you get a different result. The difference in this example is in microseconds:

```
SELECT CLOCK_TIMESTAMP() "Time 1", CLOCK_TIMESTAMP() "Time 2";
```

```
Time 1 | Time 2
2010-09-23 11:41:55.36920-04 | 2010-09-23 11:41:55.36920-04
(1 row)
```

See Also

- `STATEMENT_TIMESTAMP`
- `TRANSACTION_TIMESTAMP`

**CURRENT_DATE**

Returns the date (date-type value) on which the current transaction started.

**Behavior Type**

Stable

**Syntax**

`CURRENT_DATE()`

*Note: You can call this function without parentheses.*
Examples

<table>
<thead>
<tr>
<th>SELECT CURRENT_DATE;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-09-23</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

CURRENT_TIME

Returns a value of type TIME WITH TIMEZONE that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to CURRENT_TIME within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

CURRENT_TIME [ ( precision ) ]

Note: If you specify a column label without precision, you must also omit parentheses.

Parameters

| precision  | An integer value between 0-6, specifies to round the seconds fraction field result to the specified number of digits. |

Examples

```sql
=> SELECT CURRENT_TIME(1) AS Time;
  Time
  06:51:45.2-07
```
CURRENT_TIMESTAMP

Returns a value of type TIME WITH TIMEZONE that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to CURRENT_TIMESTAMP within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

CURRENT_TIMESTAMP ( precision )

Parameters

| precision | An integer value between 0-6, specifies to round the seconds fraction field result to the specified number of digits. |

Examples

=> SELECT CURRENT_TIMESTAMP(1) AS time;
   time
   ------------------------
   2017-03-27 06:50:49.7-07
(1 row)

=> SELECT CURRENT_TIMESTAMP(5) AS time;
   time
   --------------------------
   2017-03-27 06:50:49.69967-07
(1 row)
DATE_PART

Extracts a sub-field such as year or hour from a date/time expression, equivalent to the SQL-standard function `EXTRACT`.

Behavior Type

- Immutable if the specified date is a `TIMESTAMP`, `DATE`, or `INTERVAL`
- Stable if the specified date is a `TIMESTAMPTZ`

Syntax

```
DATE_PART ( 'field', date )
```

Parameters

<table>
<thead>
<tr>
<th><strong>field</strong></th>
<th>A constant value that specifies the sub-field to extract from <code>date</code> (see Field Values below).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>date</strong></td>
<td>The date to process, an expression that evaluates to one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>- DATE (cast to TIMESTAMP)</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>- INTERVAL</td>
</tr>
</tbody>
</table>

Field Values

<table>
<thead>
<tr>
<th><strong>CENTURY</strong></th>
<th>The century number.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0, you go from (-1) to 1.</td>
<td></td>
</tr>
<tr>
<td><strong>DAY</strong></td>
<td>The day (of the month) field (1–31).</td>
</tr>
<tr>
<td><strong>DECADE</strong></td>
<td>The year field divided by 10.</td>
</tr>
<tr>
<td><strong>DOQ</strong></td>
<td>The day within the current quarter. DOQ recognizes leap year days.</td>
</tr>
<tr>
<td><strong>DOW</strong></td>
<td>Zero-based day of the week, where Sunday=0.</td>
</tr>
<tr>
<td><strong>Note</strong>: EXTRACT's day of week numbering differs from the function <strong>TO_CHAR</strong>.</td>
<td></td>
</tr>
<tr>
<td><strong>DOY</strong></td>
<td>The day of the year (1–365/366)</td>
</tr>
<tr>
<td><strong>EPOCH</strong></td>
<td>Specifies to return one of the following:</td>
</tr>
<tr>
<td></td>
<td>• For DATE and TIMESTAMP values: the number of seconds before or since 1970-01-01 00:00:00-00 (if before, a negative number).</td>
</tr>
<tr>
<td></td>
<td>• For INTERVAL values, the total number of seconds in the interval.</td>
</tr>
<tr>
<td><strong>HOUR</strong></td>
<td>The hour field (0–23).</td>
</tr>
<tr>
<td><strong>ISODOW</strong></td>
<td>The ISO day of the week, an integer between 1 and 7 where Monday is 1.</td>
</tr>
<tr>
<td><strong>ISOWEEK</strong></td>
<td>The ISO week of the year, an integer between 1 and 53.</td>
</tr>
<tr>
<td><strong>ISOYEAR</strong></td>
<td>The ISO year.</td>
</tr>
<tr>
<td><strong>MICROSECONDS</strong></td>
<td>The seconds field, including fractional parts, multiplied by 1,000,000.</td>
</tr>
<tr>
<td></td>
<td>This includes full seconds.</td>
</tr>
<tr>
<td><strong>MILLENNIUM</strong></td>
<td>The millennium number, where the first millennium is 1 and each millenium starts on 01-01-y001. For example, millennium 2 starts on 01-01-1001.</td>
</tr>
<tr>
<td><strong>MILLISECONDS</strong></td>
<td>The seconds field, including fractional parts, multiplied by 1000. This includes full seconds.</td>
</tr>
<tr>
<td><strong>MINUTE</strong></td>
<td>The minutes field (0 - 59).</td>
</tr>
</tbody>
</table>
MONTH | For TIMESTAMP values, the number of the month within the year (1 - 12); for interval values the number of months, modulo 12 (0 - 11).
QUARTER | The calendar quarter of the specified date as an integer, where the January-March quarter is 1, valid only for TIMESTAMP values.
SECOND | The seconds field, including fractional parts, 0–59, or 0-60 if the operating system implements leap seconds.
TIME_ZONE | The time zone offset from UTC, in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.
TIMEZONE_HOUR | The hour component of the time zone offset.
TIMEZONE_MINUTE | The minute component of the time zone offset.
WEEK | The number of the week of the calendar year that the day is in.
YEAR | The year field. There is no 0 AD, so subtract BC years from AD years accordingly.

Notes
According to the ISO-8601 standard, the week starts on Monday, and the first week of a year contains January 4. Thus, an early January date can sometimes be in the week 52 or 53 of the previous calendar year. For example:

```sql
=> SELECT YEAR_ISO('01-01-2016'::DATE), WEEK_ISO('01-01-2016'), DAYOFWEEK_ISO('01-01-2016');
               YEAR_ISO | WEEK_ISO | DAYOFWEEK_ISO
-----------------+-----------+------------------
        2015 |       53 |             5
(1 row)
```

Examples

Extract the day value:

```sql
SELECT DATE_PART('DAY', TIMESTAMP '2009-02-24 20:38:40') "Day";
    Day
-----
    24
(1 row)
```

Extract the month value:
SELECT DATE_PART('MONTH', '2009-02-24 20:38:40'::TIMESTAMP) "Month";
  Month
  -------
   2
(1 row)

Extract the year value:

SELECT DATE_PART('YEAR', '2009-02-24 20:38:40'::TIMESTAMP) "Year";
  Year
  -------
   2009
(1 row)

Extract the hours:

SELECT DATE_PART('HOUR', '2009-02-24 20:38:40'::TIMESTAMP) "Hour";
  Hour
  -------
   20
(1 row)

Extract the minutes:

SELECT DATE_PART('MINUTES', '2009-02-24 20:38:40'::TIMESTAMP) "Minutes";
  Minutes
  -------
    38
(1 row)

Extract the day of quarter (DOQ):

SELECT DATE_PART('DOQ', '2009-02-24 20:38:40'::TIMESTAMP) "DOQ";
  DOQ
  -------
    55
(1 row)

**DATE**

Converts the input value to a **DATE** data type.

**Behavior Type**

- Immutable if the input value is a **TIMESTAMP**, **DATE**, **VARCHAR**, or **integer**
- Stable if the input value is a **TIMESTAMPTZ**
Syntax

```sql
DATE ( value )
```

Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>The value to convert, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• TIMESTAMP, TIMESTAMPTZ, VARCHAR, or another DATE.</td>
</tr>
<tr>
<td></td>
<td>• Integer: Vertica treats the integer as the number of days since 01/01/0001 and returns the date.</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT DATE (1);
   DATE
   0001-01-01
   (1 row)

=> SELECT DATE (734260);
   DATE
   2011-05-03
   (1 row)

=> SELECT DATE('TODAY');
   DATE
   2016-12-07
   (1 row)
```

See Also

- `TO_DATE`
- `TO_TIMESTAMP`
- `TO_TIMESTAMPTZ`
**DATE_TRUNC**

Truncates date and time values to the specified precision. The return value is the same data type as the input value. All fields that are less than the specified precision are set to 0, or to 1 for day and month.

**Behavior Type**

Stable

**Syntax**

```
DATE_TRUNC( precision, trunc-target )
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>precision</strong></th>
<th>A string constant that specifies precision for the truncated value. See Precision Field Values below. The precision must be valid for the <code>trunc-target</code> date or time.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>trunc-target</strong></td>
<td>Valid date/time expression.</td>
</tr>
</tbody>
</table>

**Precision Field Values**

<table>
<thead>
<tr>
<th><strong>MILLENNIUM</strong></th>
<th>The millennium number.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CENTURY</strong></td>
<td>The century number. The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries.</td>
</tr>
<tr>
<td><strong>DECADE</strong></td>
<td>The year field divided by 10.</td>
</tr>
<tr>
<td><strong>YEAR</strong></td>
<td>The year field. Keep in mind there is no 0 AD, so subtract BC years from AD years with care.</td>
</tr>
<tr>
<td><strong>QUARTER</strong></td>
<td>The calendar quarter of the specified date as an integer, where the</td>
</tr>
</tbody>
</table>
January-March quarter is 1.

<table>
<thead>
<tr>
<th>MONTH</th>
<th>For timestamp values, the number of the month within the year (1–12); for interval values the number of months, modulo 12 (0–11).</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEK</td>
<td>The number of the week of the year that the day is in. According to the ISO-8601 standard, the week starts on Monday, and the first week of a year contains January 4. Thus, an early January date can sometimes be in the week 52 or 53 of the previous calendar year. For example:</td>
</tr>
</tbody>
</table>
|       | ```sql
=> SELECT YEAR_ISO('01-01-2016'::DATE), WEEK_ISO('01-01-2016'), DAYOFWEEK_ISO('01-01-2016');
YEAR_ISO | WEEK_ISO | DAYOFWEEK_ISO
---------+---------+-------------
 2015 | 53 | 5
(1 row)
``` |
| DAY   | The day (of the month) field (1–31). |
| HOUR  | The hour field (0–23). |
| MINUTE| The minutes field (0–59). |
| SECOND| The seconds field, including fractional parts (0–59) (60 if leap seconds are implemented by the operating system). |
| MILLISECOND | The seconds field, including fractional parts, multiplied by 1000. Note that this includes full seconds. |
| MICROSECOND | The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds. |

**Examples**

The following example sets the field value as hour and returns the hour, truncating the minutes and seconds:

```sql
=> SELECT DATE_TRUNC('HOUR', TIMESTAMP '2012-02-24 13:38:40') AS HOUR;

 2012-02-24 13:00:00
(1 row)
```
The following example returns the year from the input `TIMESTAMPTZ '2012-02-24 13:38:40'`. The function also defaults the month and day to January 1, truncates the hour:minute:second of the timestamp, and appends the time zone (-05):

```sql
=> SELECT DATE_TRUNC('YEAR', TIMESTAMPTZ '2012-02-24 13:38:40') AS YEAR;
               YEAR
---------------------
  2012-01-01 00:00:00-05
(1 row)
```

The following example returns the year and month and defaults day of month to 1, truncating the rest of the string:

```sql
=> SELECT DATE_TRUNC('MONTH', TIMESTAMP '2012-02-24 13:38:40') AS MONTH;
               MONTH
---------------------
  2012-02-01 00:00:00
(1 row)
```

### DATEDIFF

Returns the time span between two dates, in the intervals specified. `DATEDIFF` excludes the start date in its calculation.

### Behavior Type

- Immutable if start and end dates are `TIMESTAMP, DATE, TIME, or INTERVAL`
- Stable if start and end dates are `TIMESTAMPTZ`

### Syntax

`DATEDIFF ( datepart, start, end );`

### Parameters

<table>
<thead>
<tr>
<th><code>datepart</code></th>
<th>Specifies the type of date or time intervals that <code>DATEDIFF</code> returns. If <code>datepart</code> is an expression, it must be enclosed in parentheses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>DATEDIFF((expression), start, end);</code></td>
</tr>
<tr>
<td></td>
<td><code>datepart</code> must evaluate to one of the following string literals, either quoted or unquoted:</td>
</tr>
</tbody>
</table>
- year | yy | yyyy
- quarter | qq | q
- month | mm | m
- day | dayofyear | dd | d | dy | y
- week | wk | ww
- hour | hh
- minute | mi | n
- second | ss | s
- millisecond | ms
- microsecond | mcs | us

**start, end**

Specify the start and end dates, where *start* and *end* evaluate to one of the following data types:

- `TIMESTAMP/TIMESTAMPTZ`
- `DATE`
- `TIME/TIMETZ`
- `INTERVAL`

If *end* < *start*, DATEDIFF returns a negative value.

Note: TIME and INTERVAL data types are invalid for start and end dates if *datepart* is set to year, quarter, or month.

**Compatible Start and End Date Data Types**

The following table shows which data types can be matched as start and end dates:

<table>
<thead>
<tr>
<th></th>
<th>DATE</th>
<th>TIMESTAMP</th>
<th>TIMESTAMPTZ</th>
<th>TIME</th>
<th>INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For example, if you set the start date to an INTERVAL data type, the end date must also be an INTERVAL, otherwise Vertica returns an error:

```sql
SELECT DATEDIFF(day, INTERVAL '26 days', INTERVAL '1 month ');
```

```
4
(1 row)
```

### Date Part Intervals

DATEDIFF uses the *datepart* argument to calculate the number of intervals between two dates, rather than the actual amount of time between them. DATEDIFF uses the following cutoff points to calculate those intervals:

- **year**: January 1
- **quarter**: January 1, April 1, July 1, October 1
- **month**: the first day of the month
- **week**: Sunday at midnight (24:00)

For example, if *datepart* is set to year, DATEDIFF uses January 01 to calculate the number of years between two dates. The following DATEDIFF statement sets *datepart* to year, and specifies a time span 01/01/2005 - 06/15/2008:

```sql
SELECT DATEDIFF(year, '01-01-2005':date, '12-31-2008':date);
```

```
3
(1 row)
```

DATEDIFF always excludes the start date when it calculates intervals—in this case, 01/01/2005. DATEDIFF considers only calendar year starts in its calculation, so in this case it only counts years 2006, 2007, and 2008. The function returns 3, although the actual time span is nearly four years.
If you change the start and end dates to 12/31/2004 and 01/01/2009, respectively, DATEDIFF also counts years 2005 and 2009. This time, it returns 5, although the actual time span is just over four years:

```sql
=> SELECT DATEDIFF(year, '12-31-2004'::date, '01-01-2009'::date);
       datediff
----------
           5
(1 row)
```

Similarly, DATEDIFF uses month start dates when it calculates the number of months between two dates. Thus, given the following statement, DATEDIFF counts months February through September and returns 8:

```sql
=> SELECT DATEDIFF(month, '01-31-2005'::date, '09-30-2005'::date);
       datediff
----------
           8
(1 row)
```

See Also

TIMESTAMPDIFF

DAY

Returns as an integer the day of the month from the input value.

Behavior Type

- Immutable if the input value is a TIMESTAMP, DATE, VARCHAR, or INTEGER
- Stable if the specified date is a TIMESTAMPTZ

Syntax

```sql
DAY( value )
```
Parameters

| value | The value to convert, one of the following: TIMESTAMP, TIMESTAMPTZ, INTERVAL, VARCHAR, or INTEGER. |

Examples

```sql
=> SELECT DAY (6);
   DAY
   ----
   6
   (1 row)

=> SELECT DAY(TIMESTAMP 'sep 22, 2011 12:34');
   DAY
   ----
   22
   (1 row)

=> SELECT DAY('sep 22, 2011 12:34');
   DAY
   ----
   22
   (1 row)

=> SELECT DAY(INTERVAL '35 12:34');
   DAY
   ----
   35
   (1 row)
```

DAYOFMONTH

Returns the day of the month as an integer.

Behavior Type

- Immutable if the target date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the target date is a TIMESTAMPTZ
Syntax

```
DAYOFMONTH ( date )
```

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Example

```
=> SELECT DAYOFMONTH (TIMESTAMP 'sep 22, 2011 12:34');

DAYOFMONTH
-----------
  22

(1 row)
```

DAYOFWEEK

Returns the day of the week as an integer, where Sunday is day 1.

Behavior Type

- Immutable if the target date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the target date is a TIMESTAMPTZ

Syntax

```
DAYOFWEEK ( date )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The date to process, one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT DAYOFWEEK (TIMESTAMP 'sep 17, 2011 12:34');

<table>
<thead>
<tr>
<th>DAYOFWEEK</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
```

DAYOFWEEK_ISO

Returns the ISO 8061 day of the week as an integer, where Monday is day 1.

Behavior Type

- Immutable if the target date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the target date is a TIMESTAMPTZ

Syntax

```
DAYOFWEEK_ISO ( date )
```
Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- VARCHAR</td>
<td></td>
</tr>
<tr>
<td>- DATE</td>
<td></td>
</tr>
<tr>
<td>- TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>- TIMESTAMPTZ</td>
<td></td>
</tr>
</tbody>
</table>

Examples

=> SELECT DAYOFWEEK_ISO(TIMESTAMP 'Sep 22, 2011 12:34');

<table>
<thead>
<tr>
<th>DAYOFWEEK_ISO</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

(1 row)

The following example shows how to combine the DAYOFWEEK_ISO, WEEK_ISO, and YEAR_ISO functions to find the ISO day of the week, week, and year:

=> SELECT DAYOFWEEK_ISO('Jan 1, 2000'), WEEK_ISO('Jan 1, 2000'),YEAR_ISO('Jan1,2000');

<table>
<thead>
<tr>
<th>DAYOFWEEK_ISO</th>
<th>WEEK_ISO</th>
<th>YEAR_ISO</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>52</td>
<td>1999</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- WEEK_ISO
- DAYOFWEEK_ISO

DAYOFYEAR

Returns the day of the year as an integer, where January 1 is day 1.
Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

Syntax

`DAYOFYEAR ( date )`

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT DAYOFYEAR (TIMESTAMP 'SEPT 22, 2011 12:34');
   DAYOFYEAR
----------
   265
(1 row)
```

DAYS

Returns the integer value of the specified date, where 1 AD is 1. If the date precedes 1 AD, DAYS returns a negative integer.
Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

Syntax

```
DAYS ( date )
```

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- DATE</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Example

```
=> SELECT DAYS (DATE '2011-01-22');
   DAYS
   -------
   734159
   (1 row)

=> SELECT DAYS (DATE 'March 15, 0044 BC');
   DAYS
   -------
   -15997
   (1 row)
```

EXTRACT

Retrieves sub-fields such as year or hour from date/time values and returns values of type NUMERIC. EXTRACT is intended for computational processing, rather than for formatting.
date/time values for display.

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or INTERVAL
- Stable if the specified date is a TIMESTAMPTZ

Syntax

`EXTRACT (field FROM date)`

Parameters

| field | A constant value that specifies the sub-field to extract from `date` (see Field Values below). |
| date | The date to process, an expression that evaluates to one of the following data types:  
  - `DATE` (cast to TIMESTAMP)  
  - `TIMESTAMP`  
  - `TIMESTAMPTZ`  
  - `INTERVAL` |

Field Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| CENTURY | The century number.  
The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0, you go from -1 to 1. |
<p>| DAY     | The day (of the month) field (1–31). |
| DECADE  | The year field divided by 10. |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOQ</td>
<td>The day within the current quarter. DOQ recognizes leap year days.</td>
</tr>
<tr>
<td>DOW</td>
<td>Zero-based day of the week, where Sunday=0. Note: EXTRACT's day of week numbering differs from the function TO_CHAR.</td>
</tr>
<tr>
<td>DOY</td>
<td>The day of the year (1–365/366)</td>
</tr>
</tbody>
</table>
| EPOCH    | Specifies to return one of the following:  
  - For DATE and TIMESTAMP values: the number of seconds before or since 1970-01-01 00:00:00-00 (if before, a negative number).  
  - For INTERVAL values, the total number of seconds in the interval. |
| HOUR     | The hour field (0–23). |
| ISODOW   | The ISO day of the week, an integer between 1 and 7 where Monday is 1. |
| ISOWEEK  | The ISO week of the year, an integer between 1 and 53. |
| ISOYEAR  | The ISO year. |
| MICROSECONDS | The seconds field, including fractional parts, multiplied by 1,000,000. This includes full seconds. |
| MILLENNIUM | The millennium number, where the first millennium is 1 and each millennium starts on 01-01-0001. For example, millennium 2 starts on 01-01-1001. |
| MILISECONDS | The seconds field, including fractional parts, multiplied by 1000. This includes full seconds. |
| MINUTE   | The minutes field (0 - 59). |
| MONTH    | For TIMESTAMP values, the number of the month within the year (1 - 12); for INTERVAL values the number of months, modulo 12 (0 - 11). |
| QUARTER  | The calendar quarter of the specified date as an integer, where the January-March quarter is 1, valid only for TIMESTAMP values. |
The seconds field, including fractional parts, 0–59, or 0-60 if the operating system implements leap seconds.

The time zone offset from UTC, in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.

The hour component of the time zone offset.

The minute component of the time zone offset.

The number of the week of the calendar year that the day is in.

The year field. There is no 0 AD, so subtract BC years from AD years accordingly.

Examples

Extract the day of the week and day in quarter from the current TIMESTAMP:

=> SELECT CURRENT_TIMESTAMP AS NOW;
   NOW
   --------------------------
   2016-05-03 11:36:08.829004-04
   (1 row)
=> SELECT EXTRACT (DAY FROM CURRENT_TIMESTAMP);
   date_part
   -----------
   3
   (1 row)
=> SELECT EXTRACT (DOQ FROM CURRENT_TIMESTAMP);
   date_part
   -----------
   33
   (1 row)

Extract the timezone hour from the current time:

=> SELECT CURRENT_TIMESTAMP;
   ?column?
   --------------------------
   2016-05-03 11:36:08.829004-04
   (1 row)
=> SELECT EXTRACT(TIMEZONE_HOUR FROM CURRENT_TIMESTAMP);
   date_part
   -----------
   -4
   (1 row)

Extract the number of seconds since 01-01-1970 00:00:
Extract the number of seconds between 01-01-1970 00:00 and 5 days 3 hours before:

```sql
=> SELECT EXTRACT(EPOCH FROM '-5 days 3 hours'::INTERVAL);
    date_part
--------------
   -442800.00000
(1 row)
```

Convert the results from the last example to a TIMESTAMP:

```sql
=> SELECT 'EPOCH'::TIMESTAMPTZ -442800 * '1 second'::INTERVAL;
    ?column?
--------------
1969-12-26 16:00:00-05
(1 row)
```

**GETDATE**

Returns the current statement's start date and time as a TIMESTAMP value. This function is identical to **SYSDATE**.

GETDATE uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, GETDATE converts **STATEMENT_TIMESTAMP** from TIMESTAMPTZ to TIMESTAMP.

**Behavior Type**

Stable

**Syntax**

`GETDATE()`

**Example**

```sql
=> SELECT GETDATE();
GETDATE
```
See Also

Date/Time Expressions

GETUTCDATE

Returns the current statement's start date and time as a TIMESTAMP value.

GETUTCDATE uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, GETUTCDATE converts STATEMENT_TIMESTAMP at TIME ZONE 'UTC'.

Behavior Type

Stable

Syntax

GETUTCDATE()

Example

```sql
=> SELECT GETUTCDATE();
GETUTCDATE
------------------------
2011-03-07 20:20:26.193052
(1 row)
```

See Also

- Date/Time Expressions
HOUR

Returns the hour portion of the specified date as an integer, where 0 is 00:00 to 00:59.

Behavior Type

- Immutable if the specified date is a TIMESTAMP
- Stable if the specified date is a TIMESTAMPTZ

Syntax

HOUR( date )

Parameters

date | The date to process, one of the following data types:
--- | ---
| VARCHAR
| DATE
| TIMESTAMP
| TIMESTAMPTZ
| INTERVAL

Examples

=> SELECT HOUR (TIMESTAMP 'sep 22, 2011 12:34');
HOUR
-------
 12
(1 row)
=> SELECT HOUR (INTERVAL '35 12:34');
HOUR
-------
 12
ISFINITE

Tests for the special TIMESTAMP constant INFINITY and returns a value of type BOOLEAN.

Behavior Type

Immutable

Syntax

ISFINITE (timestamp)

Parameters

<table>
<thead>
<tr>
<th>timestamp</th>
<th>Expression of type TIMESTAMP</th>
</tr>
</thead>
</table>

Examples

```
SELECT ISFINITE(TIMESTAMP '2009-02-16 21:28:30');
ISFINITE
--------
t
(1 row)
```

```
SELECT ISFINITE(TIMESTAMP 'INFINITY');
ISFINITE
--------
f
(1 row)
```
**JULIAN_DAY**

Returns the integer value of the specified day according to the Julian calendar, where day 1 is the first day of the Julian period, January 1, 4713 BC (on the Gregorian calendar, November 24, 4714 BC).

**Behavior Type**

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

**Syntax**

```
JULIAN_DAY ( date )
```

**Parameters**

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

**Example**

```
=> SELECT JULIAN_DAY (DATE 'MARCH 15, 0044 BC');
JULIAN_DAY
-----------
1705428    
(1 row)

=> SELECT JULIAN_DAY (DATE '2001-01-01');
JULIAN_DAY
-----------
```

Vertica Analytic Database (9.0.x)  Page 2418 of 6180
LAST_DAY

Returns the last day of the month in the specified date.

Behavior Type

- Immutable if the specified is a TIMESTAMP or DATE
- Stable if the specified date is a TIMESTAMPTZ

Syntax

LAST_DAY ( date )

Parameters

date | The date to process, one of the following data types:
---|---
| DATE
| TIMESTAMP
| TIMESTAMPTZ

Examples

The following example returns the last day of February as 29 because 2016 is a leap year:

```sql
=> SELECT LAST_DAY('2016-02-28 23:30 PST') "Last Day";
Last Day
----------
2016-02-29
(1 row)
```

The following example returns the last day of February in a non-leap year:

```sql
```

```sql
```
The following example returns the last day of March, after converting the string value to the specified DATE type:

```sql
=> SELECT LAST_DAY('2003/03/15') "Last";
  Last
-------------
  2012-03-31
(1 row)
```

LOCALTIME

Returns a value of type TIME that represents the start of the current transaction. The return value does not change during the transaction. Thus, multiple calls to LOCALTIME within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

```sql
LOCALTIME [ ( precision ) ]
```

Parameters

<table>
<thead>
<tr>
<th>precision</th>
<th>Rounds the result to the specified number of fractional digits in the seconds field.</th>
</tr>
</thead>
</table>

Example

```sql
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE
```
=> INSERT INTO t1 VALUES (1,2);
OUTPUT
--------
 1
(1 row)

=> SELECT LOCALTIME time;
   time
--------
 15:03:14.595296
(1 row)

=> INSERT INTO t1 VALUES (3,4);
OUTPUT
--------
 1
(1 row)

=> SELECT LOCALTIME;
   time
--------
 15:03:14.595296
(1 row)

=> COMMIT;
COMMIT
=> SELECT LOCALTIME;
   time
--------
 15:03:49.738032
(1 row)

LOCALTIMESTAMP

Returns a value of type TIMESTAMP that represents the start of the current transaction.
The return value does not change during the transaction. Thus, multiple calls to LOCALTIMESTAMP within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

LOCALTIMESTAMP [ ( precision ) ]
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision</td>
<td>Rounds the result to the specified number of fractional digits in the seconds field.</td>
</tr>
</tbody>
</table>

Example

```sql
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE

=> INSERT INTO t1 VALUES (1,2);
OUTPUT
--------
1
(1 row)

=> SELECT CURRENT_TIMESTAMP(2) timestamp;
timestamp
-----------------------------
2016-12-07 15:19:12.34-05
(1 row)

=> INSERT INTO t1 VALUES (3,4);
OUTPUT
--------
1
(1 row)

=> SELECT CURRENT_TIMESTAMP(2) timestamp;
timestamp
-----------------------------
2016-12-07 15:19:12.34-05
(1 row)

=> COMMIT;
COMMIT
=> SELECT CURRENT_TIMESTAMP(2) timestamp;
timestamp
-----------------------------
2016-12-07 15:19:13.89-05
(1 row)
```

**MICROSECOND**

Returns the microsecond portion of the specified date as an integer.
Behavior Type

- Immutable if the specified date is a "TIMESTAMP", "INTERVAL", or "VARCHAR"
- Stable if the specified date is a "TIMESTAMPTZ"

Syntax

```
MICROSECOND (date)
```

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- &quot;TIMESTAMP&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;TIMESTAMPTZ&quot;</td>
</tr>
<tr>
<td></td>
<td>- &quot;INTERVAL&quot;</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT MICROSECOND (TIMESTAMP 'Sep 22, 2011 12:34:01.123456');
MICROSECOND
-----------
123456
(1 row)
```

**MIDNIGHT_SECONDS**

Within the specified date, returns the number of seconds between midnight and the date's time portion.
Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

Syntax

MIDNIGHT_SECONDS ( date )

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- DATE</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Examples

Get the number of seconds since midnight:

```sql
=> SELECT MIDNIGHT_SECONDS(CURRENT_TIMESTAMP);
MIDNIGHT_SECONDS
-------------------
36480
(1 row)
```

Get the number of seconds between midnight and noon on March 3 2016:

```sql
=> SELECT MIDNIGHT_SECONDS('3-3-2016 12:00'::TIMESTAMP);
MIDNIGHT_SECONDS
-------------------
43200
(1 row)
```
MINUTE

Returns the minute portion of the specified date as an integer.

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, VARCHAR or INTERVAL
- Stable if the specified date is a TIMESTAMPTZ

Syntax

\[
\text{MINUTE ( } \text{date} \text{ )}
\]

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- DATE</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>- INTERVAL</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT MINUTE('12:34:03.456789');
MINUTE
-------
  34
(1 row)
=> SELECT MINUTE (TIMESTAMP 'sep 22, 2011 12:34');
MINUTE
-------
  34
```
MONTH

Returns the month portion of the specified date as an integer.

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, VARCHAR or INTERVAL
- Stable if the specified date is a TIMESTAMPTZ

Syntax

MONTH ( date )

Parameters

date | The date to process, one of the following data types:
|------------------------------------------------|
| VARCHAR
| DATE
| TIMESTAMP
| TIMESTAMPTZ
| INTERVAL

Examples

In the following examples, Vertica returns the month portion of the specified string. For example, '6-9' represent September 6.
MONTHS_BETWEEN

Returns the number of months between two dates. MONTHS_BETWEEN can return an integer or a FLOAT:

- **Integer**: The day portions of date1 and date2 are the same, and neither date is the last day of the month. MONTHS_BETWEEN also returns an integer if both dates in date1 and date2 are the last days of their respective months. For example, MONTHS_BETWEEN calculates the difference between April 30 and March 31 as 1 month.

- **FLOAT**: The day portions of date1 and date2 are different and one or both dates are not the last day of their respective months. For example, the difference between April 2 and March 1 is 1.03225806451613. To calculate month fractions, MONTHS_BETWEEN assumes all months contain 31 days.

MONTHS_BETWEEN disregards timestamp time portions.

**Behavior Type**

- Immutable if both date arguments are of data type TIMESTAMP or DATE
- Stable if either date is a TIMESTAMPTZ

**Syntax**

MONTHS_BETWEEN ( date1 , date2 );
Parameters

<table>
<thead>
<tr>
<th>date1</th>
<th>date2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the dates to evaluate where date1 and date2 evaluate to one of the following data types:</td>
<td></td>
</tr>
<tr>
<td>- DATE</td>
<td></td>
</tr>
<tr>
<td>- TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>- TIMESTAMPTZ</td>
<td></td>
</tr>
<tr>
<td>If date1 &lt; date2, MONTHS_BETWEEN returns a negative value.</td>
<td></td>
</tr>
</tbody>
</table>

Examples

Return the number of months between April 7 2016 and January 7 2015:

```sql
=> SELECT MONTHS_BETWEEN ('04-07-16'::TIMESTAMP, '01-07-15'::TIMESTAMP);
MONTHS_BETWEEN
--------
15
(1 row)
```

Return the number of months between March 31 2016 and February 28 2016 (MONTHS_BETWEEN assumes both months contain 31 days):

```sql
=> SELECT MONTHS_BETWEEN ('03-31-16'::TIMESTAMP, '02-28-16'::TIMESTAMP);
MONTHS_BETWEEN
----------
1.09677419354839
(1 row)
```

Return the number of months between March 31 2016 and February 29 2016:

```sql
=> SELECT MONTHS_BETWEEN ('03-31-16'::TIMESTAMP, '02-29-16'::TIMESTAMP);
MONTHS_BETWEEN
----------
1
(1 row)
```

NEW_TIME

Converts a timestamp value from one time zone to another and returns a TIMESTAMP.
Behavior Type

Immutable

Syntax

NEW_TIME('timestamp', 'timezone1', 'timezone2')

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>The timestamp to convert, conforms to one of the following formats:</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP/TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>- DATE</td>
</tr>
<tr>
<td></td>
<td>- Character string that can be converted to a TIMESTAMP—for example, May 24, 2012 10:00.</td>
</tr>
<tr>
<td>timezone1</td>
<td>Specify the source and target timezones, one of the strings defined in /opt/vertica/share/timezonesets. For example:</td>
</tr>
<tr>
<td>timezone2</td>
<td>- GMT: Greenwich Mean Time</td>
</tr>
<tr>
<td></td>
<td>- AST / ADT: Atlantic Standard/Daylight Time</td>
</tr>
<tr>
<td></td>
<td>- EST / EDT: Eastern Standard/Daylight Time</td>
</tr>
<tr>
<td></td>
<td>- CST / CDT: Central Standard/Daylight Time</td>
</tr>
<tr>
<td></td>
<td>- MST / MDT: Mountain Standard/Daylight Time</td>
</tr>
<tr>
<td></td>
<td>- PST / PDT: Pacific Standard/Daylight Time</td>
</tr>
</tbody>
</table>

Examples

Convert the specified time from Eastern Standard Time (EST) to Pacific Standard Time (PST):
=> SELECT NEW_TIME('05-24 13:48:00', 'EST', 'PST');

----------
2012-05-24 10:48:00
(1 row)

Convert 1:00 AM January 2012 from EST to PST:

=> SELECT NEW_TIME('01-01-12 01:00:00', 'EST', 'PST');

----------
2011-12-31 22:00:00
(1 row)

Convert the current time EST to PST:

=> SELECT NOW();

----------
2016-12-09 10:30:36.727307-05
(1 row)

=> SELECT NEW_TIME('NOW', 'EDT', 'CDT');

----------
2016-12-09 09:30:36.727307
(1 row)

The following example returns the year 45 before the Common Era in Greenwich Mean Time and converts it to Newfoundland Standard Time:

=> SELECT NEW_TIME('April 1, 45 BC', 'GMT', 'NST')::DATE;

----------
0045-03-31 BC
(1 row)

**NEXT_DAY**

Returns the date of the first instance of a particular day of the week that follows the specified date.

**Behavior Type**

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ
Syntax

NEXT_DAY( 'date', 'day-string' )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The date to process, one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
<tr>
<td>day-string</td>
<td>The day of the week to process, a CHAR or VARCHAR string or character constant. Supply the full English name such as Tuesday, or any conventional abbreviation, such as Tue or Tues. day-string is not case sensitive and trailing spaces are ignored.</td>
</tr>
</tbody>
</table>

Examples

Get the date of the first Monday that follows April 29 2016:

```sql
=> SELECT NEXT_DAY('4-29-2016'::TIMESTAMP,'Monday') "NEXT DAY" ;
   NEXT DAY
------------------
   2016-05-02
(1 row)
```

Get the first Tuesday that follows today:

```sql
SELECT NEXT_DAY(CURRENT_TIMESTAMP,'tues') "NEXT DAY" ;
   NEXT DAY
------------------
   2016-05-03
(1 row)
```
NOW [Date/Time]

Returns a value of type TIMESTAMP WITH TIME ZONE representing the start of the current transaction. NOW is equivalent to CURRENT_TIMESTAMP except that it does not accept a precision parameter.

The return value does not change during the transaction. Thus, multiple calls to CURRENT_TIMESTAMP within the same transaction return the same timestamp.

Behavior Type

Stable

Syntax

NOW()

Example

```
=> CREATE TABLE t1 (a int, b int);
CREATE TABLE
=> INSERT INTO t1 VALUES (1,2);
OUTPUT
-------
 1
(1 row)
=> SELECT NOW();
  NOW
-----------------------------
2016-12-09 13:00:08.74685-05
(1 row)
=> INSERT INTO t1 VALUES (3,4);
OUTPUT
-------
 1
(1 row)
=> SELECT NOW();
  NOW
-----------------------------
2016-12-09 13:00:08.74685-05
(1 row)
=> COMMIT;
```
OVERLAPS

Evaluates two time periods and returns true when they overlap, false otherwise.

Behavior Type

- Stable when TIMESTAMP and TIMESTAMPTZ are both used, or when TIMESTAMPTZ is used with INTERVAL
- Immutable otherwise

Syntax

( start, end ) OVERLAPS ( start, end )
( start, interval) OVERLAPS ( start, interval )

Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>DATE, TIME, or TIMESTAMP/TIMESTAMPTZ value that specifies the beginning of a time period.</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>DATE, TIME, or TIMESTAMP/TIMESTAMPTZ value that specifies the end of a time period.</td>
</tr>
<tr>
<td>interval</td>
<td>Value that specifies the length of the time period.</td>
</tr>
</tbody>
</table>

Examples

Evaluate whether date ranges Feb 16 - Dec 21, 2016 and Oct 10 2008 - Oct 3 2016 overlap:

=> SELECT (DATE '2016-02-16', DATE '2016-12-21') OVERLAPS (DATE '2008-10-30', DATE '2016-10-30');
overlaps
Evaluate whether date ranges Feb 16 - Dec 21, 2016 and Jan 01 - Oct 30 2008 - Oct 3, 2016 overlap:

```sql
=> SELECT (DATE '2016-02-16', DATE '2016-12-21') OVERLAPS (DATE '2008-01-30', DATE '2008-10-30');
(1 row)
```

Evaluate whether date range Feb 02 2016 + 1 week overlaps with date range Oct 16 2016 - 8 months:

```sql
=> SELECT (DATE '2016-02-16', INTERVAL '1 week') OVERLAPS (DATE '2016-10-16', INTERVAL '-8 months');
(1 row)
```

**QUARTER**

Returns calendar quarter of the specified date as an integer, where the January-March quarter is 1.

**Syntax**

`QUARTER ( date )`

**Behavior Type**

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR.
- Stable if the specified date is a TIMESTAMPTZ

**Parameters**

<table>
<thead>
<tr>
<th><code>date</code></th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
</tbody>
</table>
Examples

```sql
=> SELECT QUARTER (TIMESTAMP 'sep 22, 2011 12:34');
QUARTER
--------
3
(1 row)
```

ROUND

Rounds the specified date or time. If you omit the precision argument, ROUND rounds to day (DD) precision.

Behavior Type

- Immutable if the target date is a TIMESTAMP or DATE
- Stable if the target date is a TIMESTAMPTZ

Syntax

```sql
ROUND( rounding-target[, 'precision'] )
```

Parameters

<table>
<thead>
<tr>
<th>rounding-target</th>
<th>An expression that evaluates to one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DATE</td>
</tr>
<tr>
<td>Date</td>
<td>TIMESTAMP/TIMESTAMPTZ</td>
</tr>
<tr>
<td>Date</td>
<td>TIMESTAMPTZ</td>
</tr>
</tbody>
</table>
### precision

A string constant that specifies precision for the rounded value, one of the following:

- **Century**: CC | SCC
- **Year**: SYYY | YYYY | YEAR | YYY | YY | Y
- **ISO Year**: IYYY | IYY | IY | I
- **Quarter**: Q
- **Month**: MONTH | MON | MM | RM
- **Same weekday as first day of year**: WW
- **Same weekday as first day of ISO year**: IW
- **Same weekday as first day of month**: W
- **Day** (default): DDD | DD | J
- **First weekday**: DAY | DY | D
- **Hour**: HH | HH12 | HH24
- **Minute**: MI
- **Second**: SS

**Note**: Hour, minute, and second rounding is not supported by DATE expressions.

### Examples

Round to the nearest hour:

```sql
=> SELECT ROUND(CURRENT_TIMESTAMP, 'HH');

ROUND
2016-04-28 15:00:00
(1 row)
```

Round to the nearest month:
See Also

**TIMESTAMP_ROUND**

**SECOND**

Returns the seconds portion of the specified date as an integer.

**Syntax**

```sql
SECOND ( date )
```

**Behavior Type**

Immutable, except for TIMESTAMPTZ arguments where it is stable.

**Parameters**

| `date` | The date to process, one of the following data types:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>- INTERVAL</td>
</tr>
</tbody>
</table>
Examples

=> SELECT SECOND ('23:34:03.456789');
SECOND
-----
3
(1 row)

=> SELECT SECOND (TIMESTAMP 'sep 22, 2011 12:34');
SECOND
-----
0
(1 row)

=> SELECT SECOND (INTERVAL '35 12:34:03.456789');
SECOND
-----
3
(1 row)

STATEMENT_TIMESTAMP

Similar to TRANSACTION_TIMESTAMP, returns a value of type TIMESTAMP WITH TIME ZONE that represents the start of the current statement.

The return value does not change during statement execution. Thus, different stages of statement execution always have the same timestamp.

Behavior Type

Stable

Syntax

STATEMENT_TIMESTAMP()

Example

=> SELECT foo, bar FROM (SELECT STATEMENT_TIMESTAMP() AS foo, (SELECT STATEMENT_TIMESTAMP() as bar)bar);

<table>
<thead>
<tr>
<th>foo</th>
<th>bar</th>
</tr>
</thead>
</table>
(1 row)
See Also

- **CLOCK_TIMESTAMP**
- **TRANSACTION_TIMESTAMP**

**SYSDATE**

Returns the current statement's start date and time as a TIMESTAMP value. This function is identical to **GETDATE**.

SYSDATE uses the date and time supplied by the operating system on the server to which you are connected, which is the same across all servers. Internally, GETDATE converts **STATEMENT_TIMESTAMP** from TIMESTAMPTZ to TIMESTAMP.

**Behavior Type**

Stable

**Syntax**

```sql
SYSDATE()
```

**Note:** You can call this function with no parentheses.

**Example**

```sql
=> SELECT SYSDATE;
  sysdate
-------------------
  2016-12-12 06:11:10.699642
(1 row)
```

**See Also**

Date/Time Expressions
TIME_SLICE

Aggregates data by different fixed-time intervals and returns a rounded-up input TIMESTAMP value to a value that corresponds with the start or end of the time slice interval.

Given an input TIMESTAMP value such as 2000-10-28 00:00:01, the start time of a 3-second time slice interval is 2000-10-28 00:00:00, and the end time of the same time slice is 2000-10-28 00:00:03.

Behavior Type

Immutable

Syntax

```
TIME_SLICE( expression, slice-length [, 'time-unit' [, 'start-or-end' ] ] )
```

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>One of the following:</td>
</tr>
<tr>
<td>slice-length</td>
<td>A positive integer that specifies the slice length.</td>
</tr>
<tr>
<td>time-unit</td>
<td>Time unit of the slice, one of the following:</td>
</tr>
</tbody>
</table>

- Column of type TIMESTAMP
- String constant that can be parsed into a TIMESTAMP value. For example:

  '2004-10-19 10:23:54'

Vertica evaluates `expression` on each row.
**Null Argument Handling**

TIME_SLICE handles null arguments as follows:

- TIME_SLICE returns an error when any one of slice-length, time-unit, or start-or-end parameters is null.
- If expression is null and slice-length, time-unit, or start-or-end contain legal values, TIME_SLICE returns a NULL value instead of an error.

**Usage**

The following command returns the (default) start time of a 3-second time slice:

```sql
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3);

TIME_SLICE
------------------
2009-09-19 00:00:00
(1 row)
```

The following command returns the end time of a 3-second time slice:

```sql
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'SECOND', 'END');

TIME_SLICE
------------------
2009-09-19 00:00:03
(1 row)
```

This command returns results in milliseconds, using a 3-second time slice:
This command returns results in microseconds, using a 9-second time slice:

```sql
=> SELECT TIME_SLICE('2009-09-19 00:00:01', 3, 'us');

TIME_SLICE
------------------------
2009-09-19 00:00:00.999999

(1 row)
```

The next example uses a 3-second interval with an input value of '00:00:01'. To focus specifically on seconds, the example omits date, though all values are implied as being part of the timestamp with a given input of '00:00:01':

- '00:00:00' is the start of the 3-second time slice
- '00:00:03' is the end of the 3-second time slice.
- '00:00:03' is also the start of the second 3-second time slice. In time slice boundaries, the end value of a time slice does not belong to that time slice; it starts the next one.

When the time slice interval is not a factor of 60 seconds, such as a given slice length of 9 in the following example, the slice does not always start or end on 00 seconds:

```sql
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 9);

TIME_SLICE
------------------------
2009-02-14 20:12:54

(1 row)
```

This is expected behavior, as the following properties are true for all time slices:

- Equal in length
- Consecutive (no gaps between them)
- Non-overlapping
To force the above example ('2009-02-14 20:13:01') to start at '2009-02-14 20:13:00', adjust the output timestamp values so that the remainder of 54 counts up to 60:

```sql
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 9) + '6 seconds'::INTERVAL AS time;

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-14 20:13:00</td>
</tr>
</tbody>
</table>

(1 row)
```

Alternatively, you could use a different slice length, which is divisible by 60, such as 5:

```sql
=> SELECT TIME_SLICE('2009-02-14 20:13:01', 5);

<table>
<thead>
<tr>
<th>TIME_SLICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-14 20:13:00</td>
</tr>
</tbody>
</table>

(1 row)
```

A TIMESTAMPTZ value is implicitly cast to TIMESTAMP. For example, the following two statements have the same effect.

```sql
=> SELECT TIME_SLICE('2009-09-23 11:12:01'::TIMESTAMPTZ, 3);

<table>
<thead>
<tr>
<th>TIME_SLICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-09-23 11:12:00</td>
</tr>
</tbody>
</table>

(1 row)
```

```sql
=> SELECT TIME_SLICE('2009-09-23 11:12:01'::TIMESTAMP::TIMESTAMP, 3);

<table>
<thead>
<tr>
<th>TIME_SLICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-09-23 11:12:00</td>
</tr>
</tbody>
</table>

(1 row)
```

**Examples**

You can use the SQL analytic functions FIRST_VALUE and LAST_VALUE to find the first/last price within each time slice group (set of rows belonging to the same time slice). This structure could be useful if you want to sample input data by choosing one row from each time slice group.
=> SELECT date_key, transaction_time, sales_dollar_amount, TIME_SLICE(DATE '2000-01-01' + date_key + transaction_time, 3), FIRST_VALUE(sales_dollar_amount) OVER (PARTITION BY TIME_SLICE(DATE '2000-01-01' + date_key + transaction_time, 3) ORDER BY DATE '2000-01-01' + date_key + transaction_time) AS first_value FROM store.store_sales_fact LIMIT 20;

<table>
<thead>
<tr>
<th>date_key</th>
<th>transaction_time</th>
<th>sales_dollar_amount</th>
<th>time_slice</th>
<th>first_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:41:16</td>
<td>164</td>
<td>2000-01-02 00:41:15</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>00:41:33</td>
<td>310</td>
<td>2000-01-02 00:41:33</td>
<td>310</td>
<td></td>
</tr>
<tr>
<td>15:33:15</td>
<td>419</td>
<td>2000-01-02 15:33:15</td>
<td>419</td>
<td></td>
</tr>
<tr>
<td>16:36:29</td>
<td>466</td>
<td>2000-01-02 16:36:27</td>
<td>466</td>
<td></td>
</tr>
<tr>
<td>16:36:44</td>
<td>250</td>
<td>2000-01-02 16:36:42</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>03:11:28</td>
<td>39</td>
<td>2000-01-03 03:11:27</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>11:58:05</td>
<td>369</td>
<td>2000-01-04 11:58:03</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td>11:58:52</td>
<td>449</td>
<td>2000-01-04 11:58:51</td>
<td>449</td>
<td></td>
</tr>
<tr>
<td>19:01:21</td>
<td>201</td>
<td>2000-01-04 19:01:21</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>22:15:05</td>
<td>156</td>
<td>2000-01-04 22:15:03</td>
<td>156</td>
<td></td>
</tr>
</tbody>
</table>

TIME_SLICE rounds the transaction time to the 3-second slice length.

The following example uses the analytic (window) OVER clause to return the last trading price (the last row ordered by TickTime) in each 3-second time slice partition:

=> SELECT DISTINCT TIME_SLICE(TickTime, 3), LAST_VALUE(price) OVER (PARTITION BY TIME_SLICE(TickTime, 3) ORDER BY TickTime ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING);

Note: If you omit the windowing clause from an analytic clause, LAST_VALUE defaults to RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. Results can seem non-intuitive, because instead of returning the value from the bottom of the current partition, the function returns the bottom of the window, which continues to change along with the current input row that is being processed. For more information, see Time Series Analytics and SQL Analytics in Analyzing Data.

In the next example, FIRST_VALUE is evaluated once for each input record and the data is sorted by ascending values. Use SELECT DISTINCT to remove the duplicates and return only one output record per TIME_SLICE:
The information output by the above query can also return MIN, MAX, and AVG of the trading prices within each time slice.

```
=> SELECT DISTINCT TIME_SLICE(TickTime, 3), FIRST_VALUE(Price) OVER (PARTITION BY TIME_SLICE(TickTime, 3))
ORDER BY TickTime ASC
FROM tick_store;
```

<table>
<thead>
<tr>
<th>TIME_SLICE</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-09-21 00:00:06</td>
<td>20.00</td>
</tr>
<tr>
<td>2009-09-21 00:00:09</td>
<td>30.00</td>
</tr>
<tr>
<td>2009-09-21 00:00:00</td>
<td>10.00</td>
</tr>
</tbody>
</table>

(3 rows)

**See Also**

- [Aggregate Functions](#)
- [FIRST_VALUE [Analytic]](#)
- [LAST_VALUE [Analytic]](#)
- [TIMESERIES Clause](#)
- [TS_FIRST_VALUE](#)
- [TS_LAST_VALUE](#)
- [Using Time Zones With Vertica](#)

**TIMEOFDAY**

Returns the wall-clock time as a text string. Function results advance during transactions.

**Behavior Type**

Volatile
Syntax

TIMEOFDAY()

Example

=> SELECT TIMEOFDAY();

TIMEOFDAY
Mon Dec 12 08:18:01.022710 2016 EST
(1 row)

TIMESTAMPADD

Adds the specified number of intervals to a TIMESTAMP or TIMESTAMPTZ value and returns a result of the same data type.

Behavior Type

- Immutable if the input date is a TIMESTAMP
- Stable if the input date is a TIMESTAMPTZ

Syntax

TIMESTAMPADD( datepart, count, start-date );

Parameters

<table>
<thead>
<tr>
<th>datepart</th>
<th>Specifies the type of time intervals that TIMESTAMPADD adds to the specified start date. If datepart is an expression, it must be enclosed in parentheses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TIMESTAMPADD((expression), interval, start);</td>
</tr>
<tr>
<td></td>
<td>datepart must evaluate to one of the following string literals, either quoted or unquoted:</td>
</tr>
</tbody>
</table>
### Examples

Add two months to the current date:

```sql
=> SELECT CURRENT_TIMESTAMP AS Today;
    Today
-----------------------------
  2016-05-02 06:56:57.923045-04
(1 row)

=> SELECT TIMESTAMPPADD (MONTH, 2, (CURRENT_TIMESTAMP)) AS TodayPlusTwoMonths;
   TodayPlusTwoMonths
-----------------------------
  2016-07-02 06:56:57.923045-04
(1 row)
```

Add 14 days to the beginning of the current month:

```sql
=> SELECT TIMESTAMPPADD (DD, 14, (SELECT TRUNC((CURRENT_TIMESTAMP), 'MM')));
    timestampadd
-----------------------------
  2016-05-15 00:00:00
(1 row)
```
TIMESTAMPDIFF

Returns the time span between two TIMESTAMP or TIMESTAMPTZ values, in the intervals specified. TIMESTAMPDIFF excludes the start date in its calculation.

Behavior Type

- Immutable if start and end dates are TIMESTAMP
- Stable if start and end dates are TIMESTAMPTZ

Syntax

TIMESTAMPDIFF (datepart, start, end);

Parameters

| datepart | Specifies the type of date or time intervals that TIMESTAMPDIFF returns. If datepart is an expression, it must be enclosed in parentheses: TIMESTAMPDIFF((expression), start, end);
| datepart | must evaluate to one of the following string literals, either quoted or unquoted:
| year | yy | yyyy
| quarter | qq | q
| month | mm | m
| day | dayofyear | dd | d | dy | y
| week | wk | ww
| hour | hh
| minute | mi | n
| second | ss | s
### Date Part Intervals

TIMESTAMPDIFF uses the `datepart` argument to calculate the number of intervals between two dates, rather than the actual amount of time between them. For detailed information, see `DATEDIFF`.

### Examples

```sql
=> SELECT TIMESTAMPDIFF (YEAR, '1-1-2006 12:34:00', '1-1-2008 12:34:00');
timestampdiff
-----------------
  2
(1 row)
```

### See Also

`DATEDIFF`

### TIMESTAMP_ROUND

Rounds the specified TIMESTAMP. If you omit the precision argument, `TIMESTAMP_ROUND` rounds to day (DD) precision.
Behavior Type

- Immutable if the target date is a TIMESTAMP
- Stable if the target date is a TIMESTAMPTZ

Syntax

```
TIMESTAMP_ROUND ( rounding-target[, 'precision'] )
```

Parameters

<table>
<thead>
<tr>
<th>rounding-target</th>
<th>An expression that evaluates to one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• TIMESTAMP/TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>precision</th>
<th>A string constant that specifies precision for the rounded value, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Century: CC</td>
</tr>
<tr>
<td></td>
<td>• Year: SYYY</td>
</tr>
<tr>
<td></td>
<td>• ISO Year: IYYY</td>
</tr>
<tr>
<td></td>
<td>• Quarter: Q</td>
</tr>
<tr>
<td></td>
<td>• Month: MONTH</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of year: Ww</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of ISO year: IW</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of month: W</td>
</tr>
<tr>
<td></td>
<td>• Day (default): DDD</td>
</tr>
<tr>
<td></td>
<td>• First weekday: DAY</td>
</tr>
</tbody>
</table>
Examples

Round to the nearest hour:

```sql
=> SELECT TIMESTAMP_ROUND(CURRENT_TIMESTAMP, 'HH');
   ROUND
---------------------
 2016-04-28 15:00:00
(1 row)
```

Round to the nearest month:

```sql
=> SELECT TIMESTAMP_ROUND('9-22-2011 12:34:00'::TIMESTAMP, 'MM');
   ROUND
---------------------
 2011-10-01 00:00:00
(1 row)
```

See Also

ROUND

TIMESTAMP_TRUNC

Truncates the specified TIMESTAMP. If you omit the precision argument, TIMESTAMP_TRUNC truncates to day (DD) precision.

Behavior Type

- Immutable if the target date is a TIMESTAMP
- Stable if the target date is a TIMESTAMPTZ
Syntax

```
TIMESTAMP_TRUNC( trunc-target[, 'precision'] )
```

Parameters

<table>
<thead>
<tr>
<th><strong>trunc-target</strong></th>
<th>An expression that evaluates to one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <code>TIMESTAMP/TIMESTAMPZ</code></td>
</tr>
<tr>
<td></td>
<td>• <code>TIMESTAMPZ</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>precision</strong></th>
<th>A string constant that specifies precision for the truncated value, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Century: CC</td>
</tr>
<tr>
<td></td>
<td>• Year: SYYY</td>
</tr>
<tr>
<td></td>
<td>• ISO Year: IYYYY</td>
</tr>
<tr>
<td></td>
<td>• Quarter: Q</td>
</tr>
<tr>
<td></td>
<td>• Month: MONTH</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of year: WW</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of ISO year: IW</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of month: W</td>
</tr>
<tr>
<td></td>
<td>• Day: DDD</td>
</tr>
<tr>
<td></td>
<td>• First weekday: DAY</td>
</tr>
<tr>
<td></td>
<td>• Hour: HH</td>
</tr>
<tr>
<td></td>
<td>• Minute: MI</td>
</tr>
<tr>
<td></td>
<td>• Second: SS</td>
</tr>
</tbody>
</table>

**Note:** Hour, minute, and second truncating is not supported by Vertica.
DATE expressions.

Examples

Truncate to the current hour:

```
=> SELECT TIMESTAMP_TRUNC(CURRENT_TIMESTAMP, 'HH');
TIMESTAMP_TRUNC
---------------------
2016-04-29 08:00:00
(1 row)
```

Truncate to the month:

```
=> SELECT TIMESTAMP_TRUNC('9-22-2011 12:34:00'::TIMESTAMP, 'MM');
TIMESTAMP_TRUNC
---------------------
2011-09-01 00:00:00
(1 row)
```

See Also

TRUNC

TRANSACTION_TIMESTAMP

Returns a value of type TIME WITH TIMEZONE that represents the start of the current transaction.

The return value does not change during the transaction. Thus, multiple calls to TRANSACTION_TIMESTAMP within the same transaction return the same timestamp. TRANSACTION_TIMESTAMP is equivalent to CURRENT_TIMESTAMP, except it does not accept a precision parameter.

Behavior Type

Stable
Syntax

TRANSACTION_TIMESTAMP()

Example

```sql
=> SELECT foo, bar FROM (SELECT TRANSACTION_TIMESTAMP() AS foo)foo, (SELECT TRANSACTION_TIMESTAMP() as bar)bar;

foo | bar
--- | ---
2016-12-12 08:18:00.988528-05 | 2016-12-12 08:18:00.988528-05

(1 row)
```

See Also

- `CLOCK_TIMESTAMP`
- `STATEMENT_TIMESTAMP`

TRUNC

Truncates the specified date or time. If you omit the precision argument, TRUNC truncates to day (DD) precision.

Behavior Type

- Immutable if the target date is a `TIMESTAMP` or `DATE`
- Stable if the target date is a `TIMESTAMPZ`

Syntax

```sql
TRUNC( trunc-target[, 'precision'] )
```
## Parameters

<table>
<thead>
<tr>
<th><strong>trunc-target</strong></th>
<th>An expression that evaluates to one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP/TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>precision</strong></th>
<th>A string constant that specifies precision for the truncated value, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Century: CC</td>
</tr>
<tr>
<td></td>
<td>• Year: SYYY</td>
</tr>
<tr>
<td></td>
<td>• ISO Year: IYYY</td>
</tr>
<tr>
<td></td>
<td>• Quarter: Q</td>
</tr>
<tr>
<td></td>
<td>• Month: MONTH</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of year: WW</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of ISO year: IW</td>
</tr>
<tr>
<td></td>
<td>• Same weekday as first day of month: W</td>
</tr>
<tr>
<td></td>
<td>• Day (default): DDD</td>
</tr>
<tr>
<td></td>
<td>• First weekday: DAY</td>
</tr>
<tr>
<td></td>
<td>• Hour: HH</td>
</tr>
<tr>
<td></td>
<td>• Minute: MI</td>
</tr>
<tr>
<td></td>
<td>• Second: SS</td>
</tr>
</tbody>
</table>

**Note:** Hour, minute, and second truncating is not supported by DATE expressions.
Examples

Truncate to the current hour:

```sql
=> SELECT TRUNC(CURRENT_TIMESTAMP, 'HH');
TRUNC
------------------------
2016-04-29 10:00:00
(1 row)
```

Truncate to the month:

```sql
=> SELECT TRUNC('9-22-2011 12:34:00':TIMESTAMP, 'MM');
TIMESTAMP_TRUNC
------------------------
2011-09-01 00:00:00
(1 row)
```

See Also

TIMESTAMP_TRUNC

WEEK

Returns the week of the year for the specified date as an integer, where the first week begins on the first Sunday on or preceding January 1.

Syntax

```sql
WEEK ( date )
```

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The date to process, one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Examples

January 2 is on Saturday, so WEEK returns 1:

```sql
=> SELECT WEEK ('1-2-2016'::DATE);
WEEK
------
  1
(1 row)
```

January 3 is the second Sunday in 2016, so WEEK returns 2:

```sql
=> SELECT WEEK ('1-3-2016'::DATE);
WEEK
------
  2
(1 row)
```

WEEK_ISO

Returns the week of the year for the specified date as an integer, where the first week starts on Monday and contains January 4. This function conforms with the ISO 8061 standard.

Syntax

`WEEK_ISO ( date )`
Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>The date to process, one of the following data types:</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMPTZ</td>
</tr>
</tbody>
</table>

Examples

The first week of 2016 begins on Monday January 4:

```sql
=> SELECT WEEK_ISO ('1-4-2016':::DATE);
WEEK_ISO
-------
 1
(1 row)
```

January 3 2016 returns week 53 of the previous year (2015):

```sql
=> SELECT WEEK_ISO ('1-3-2016':::DATE);
WEEK_ISO
-------
 53
(1 row)
```

In 2015, January 4 is on Sunday, so the first week of 2015 begins on the preceding Monday (December 29 2014):

```sql
=> SELECT WEEK_ISO ('12-29-2014':::DATE);
WEEK_ISO
-------
 1
(1 row)
```
YEAR

Returns an integer that represents the year portion of the specified date.

Syntax

YEAR( date )

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, VARCHAR, or INTERVAL
- Stable if the specified date is a TIMESTAMPTZ

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• DATE</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPTZ</td>
</tr>
<tr>
<td></td>
<td>• INTERVAL</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT YEAR(CURRENT_DATE::DATE);
YEAR
-------
2016
2016
(1 row)
```
See Also

YEAR_ISO

YEAR_ISO

Returns an integer that represents the year portion of the specified date. The return value is based on the ISO 8061 standard.

The first week of the ISO year is the week that contains January 4.

Syntax

YEAR_ISO ( date )

Behavior Type

- Immutable if the specified date is a TIMESTAMP, DATE, or VARCHAR
- Stable if the specified date is a TIMESTAMPTZ

Parameters

<table>
<thead>
<tr>
<th>date</th>
<th>The date to process, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- DATE</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>- TIMESTAMPTZ</td>
</tr>
</tbody>
</table>
Examples

> SELECT YEAR_ISO(CURRENT_DATE::DATE);
> YEAR_ISO
> 2016
> (1 row)

See Also

YEAR

**Error-handling Functions**

Error-handling functions take a string and return the string when the query is executed.

**THROW_ERROR**

Returns a user-defined error message.

In a multi-node cluster, race conditions might cause the order of error messages to differ.

**Syntax**

THROW_ERROR ( message )

**Parameters**

- **message**: The VARCHAR string to return.

**Examples**

Return an error message when a CASE statement is met:
CREATE TABLE pitcher_err (some_text varchar);
CREATE TABLE
=> COPY pitcher_err FROM STDIN;
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.
=> big foo value
=> bigger foo other value
=> bar another foo value
=> 
=> SELECT (CASE WHEN true THEN THROW_ERROR('Failure!!!') ELSE some_text END) FROM pitcher_err;
ERROR 7137: USER GENERATED ERROR: Failure!!!

Return an error message when a CASE statement using REGEXP_LIKE is met:

=> SELECT (CASE WHEN REGEXP_LIKE(some_text, 'other') THEN THROW_ERROR('Failure at "'|| some_text || '"') END) FROM pitcher_err;
ERROR 4566: USER GENERATED ERROR: Failure at "bar another foo value"
Formatting Functions

Formatting functions provide a powerful tool set for converting various data types (DATE/TIME, INTEGER, FLOATING POINT) to formatted strings and for converting from formatted strings to specific data types.

TO_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format. This function is the inverse of BITSTRING_TO_BINARY.

Behavior Type

Immutable

Syntax

TO_BITSTRING ( expression )

Parameters

| expression | The VARCHAR string to process. |

Examples

=> SELECT TO_BITSTRING('ab'::BINARY(2));
   to_bitstring
   ---------------------
   0110000101100010
   (1 row)

=> SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
   to_bitstring
   --------------
   00010000
   (1 row)
SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
to_bitstring
--------------
11110000
(1 row)

See Also

BITCOUNT

TO_CHAR

Converts various date/time and numeric values into text strings.

Behavior Type

Stable

Syntax

TO_CHAR ( expression [, pattern ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Specifies the value to convert, one of the following data types:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• DOUBLE PRECISION</td>
</tr>
<tr>
<td></td>
<td>• INTEGER</td>
</tr>
<tr>
<td></td>
<td>• INTERVAL</td>
</tr>
<tr>
<td></td>
<td>• TIME/TIMETZ</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMP/TIMESTAMPTZ</td>
</tr>
<tr>
<td>pattern</td>
<td>A CHAR or VARCHAR that specifies an output pattern string. See:</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Template Patterns for Date/Time Formatting</td>
</tr>
</tbody>
</table>
Notes

- **TO_CHAR(any)** casts any type, except BINARY/VARBINARY, to VARCHAR.

  The following example returns an error if you try to cast TO_CHAR to a binary data type:

  ```sql
  => SELECT TO_CHAR('abc'::VARBINARY);
  ERROR:  cannot cast type varbinary to varchar
  ```

- **TO_CHAR** accepts TIME and TIMETZ data types as inputs if you explicitly cast TIME to TIMESTAMP and TIMETZ to TIMESTAMPTZ.

  ```sql
  => SELECT TO_CHAR(TIME '14:34:06.4','HH12:MI am');
  => SELECT TO_CHAR(TIMETZ '14:34:06.4+6','HH12:MI am');
  ```

  You can extract the timezone hour from TIMETZ:

  ```sql
  => SELECT EXTRACT(timezone_hour FROM TIMETZ '10:30+13:30');
  date_part
  13
  (1 row)
  ```

- Ordinary text is allowed in to_char templates and is output literally. You can put a substring in double quotes to force it to be interpreted as literal text even if it contains pattern key words. For example, in "Hello Year YYYY", the YYYY is replaced by the year data, but the single Y in Year is not.

- **TO_CHAR**'s day-of-the-week numbering (see the 'D' template pattern) is different from that of the EXTRACT function.

- Given an INTERVAL type, TO_CHAR formats HH and HH12 as hours in a single day, while HH24 can output hours exceeding a single day, for example, >24.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: '"YYYY Month\\"'

- **TO_CHAR** does not support the use of V combined with a decimal point. For example: 99.9V99 is not allowed.
- When rounding, the last digit of the rounded representation is selected to be even if the number is exactly half way between the two.

## Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT TO_CHAR(CURRENT_TIMESTAMP, 'Day, DD HH12:MI:SS');</td>
<td>'Tuesday, 06 05:39:18'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(CURRENT_TIMESTAMP, 'FMDay, FMDD HH12:MI:SS');</td>
<td>'Tuesday, 6 05:39:18'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(TIMetz '14:34:06.4+6','HH12:MI am'); TO_CHAR</td>
<td>04:34 am</td>
</tr>
<tr>
<td>SELECT TO_CHAR(-0.1, '99.99');</td>
<td>'-.10'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(-0.1, 'FM9.99');</td>
<td>'-.1'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(0.1, '0.9');</td>
<td>'0.1'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(12, '9990999.9');</td>
<td>'0012.0'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(12, 'FM9990999.9');</td>
<td>'0012.'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, '999');</td>
<td>'485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(-485, '999');</td>
<td>'-485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, '9 9 9');</td>
<td>'4 8 5'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(1485, '9,999');</td>
<td>'1 485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(1485, '96999');</td>
<td>'1 485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(148.5, '999.999');</td>
<td>'148.500'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(148.5, 'FM999.999');</td>
<td>'148.5'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(148.5, 'FM999.990');</td>
<td>'148.500'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(148.5, '999D999');</td>
<td>'148,500'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(3148.5, '9G999D999');</td>
<td>'3 148,500'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(-485, '999S');</td>
<td>'485-'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(-485, '999MI');</td>
<td>'485-'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, '999MI');</td>
<td>'485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, 'FM999MI');</td>
<td>'485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, 'PL999');</td>
<td>'+485'</td>
</tr>
<tr>
<td>SELECT TO_CHAR(485, 'SG999');</td>
<td>'+485'</td>
</tr>
</tbody>
</table>
TO_DATE

Converts a string value to a DATE type.

**Behavior Type**

Stable

**Syntax**

```
TO_DATE ( expression , pattern )
```
Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Specifies the string value to convert, either CHAR or VARCHAR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A CHAR or VARCHAR that specifies an output pattern string. See:</td>
</tr>
<tr>
<td></td>
<td>• Template Patterns for Date/Time Formatting</td>
</tr>
<tr>
<td></td>
<td>• Template Patterns for Numeric Formatting</td>
</tr>
</tbody>
</table>

Input Value Considerations

TO_DATE requires a CHAR or VARCHAR expression. For other input types, use TO_CHAR to perform an explicit cast to a CHAR or VARCHAR before using this function.

Notes

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: "\"YYYY Month\""
- TO_TIMESTAMP, TO_TIMESTAMP_TZ, and TO_DATE skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:
  - TO_TIMESTAMP('2000 JUN', 'YYYY MON') is correct.
  - TO_TIMESTAMP('2000 JUN', 'FXYYYY MON') returns an error, because TO_TIMESTAMP expects one space only.
- The YYYY conversion from string to TIMESTAMP or DATE has a restriction if you use a year with more than four digits. You must use a non-digit character or template after YYYY, otherwise the year is always interpreted as four digits. For example, given the following arguments, TO_DATE interprets the five-digit year 20000 as a four-digit year:

```
=> SELECT TO_DATE('200001131', 'YYYYMMDD');
TO_DATE
-----------
```

Instead, use a non-digit separator after the year. For example:

```
=> SELECT TO_DATE('2000-1131', 'YYYY-MMDD');
           TO_DATE
-------------
       2000-12-01
          (1 row)
```

- In conversions from string to TIMESTAMP or DATE, the CC field is ignored if there is a YY, YYYY or Y,YYY field. If CC is used with YY or Y, then the year is computed as (CC–1)*100+YY.

**Examples**

```
=> SELECT TO_DATE('13 Feb 2000', 'DD Mon YYYY');
          to_date
-------------
      2000-02-13
         (1 row)
```

**See Also**

Date/Time Functions

**TO_HEX**

Returns a VARCHAR or VARBINARY representing the hexadecimal equivalent of a number. This function is the inverse of HEX_TO_BINARY.

**Behavior Type**

Immutable

**Syntax**

```
TO_HEX ( number )
```
Parameters

| number   | An INTEGER or VARBINARY value to convert to hexadecimal. If you supply a VARBINARY argument, the function's return value is not preceded by 0x. |

Examples

=> SELECT TO_HEX(123456789);
```
TO_HEX
-------
75bcd15
(1 row)
```

For VARBINARY inputs, the returned value is not preceded by 0x. For example:

=> SELECT TO_HEX('ab':binary(2));
```
TO_HEX
-------
6162
(1 row)
```

TO_TIMESTAMP

Converts a string value or a UNIX/POSIX epoch value to a TIMESTAMP type.

Behavior Type

Stable

Syntax

TO_TIMESTAMP ( expression, pattern )
TO_TIMESTAMP ( unix-epoch )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Specifies the string value to convert, either CHAR or VARCHAR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A CHAR or VARCHAR that specifies an output pattern string. See:</td>
</tr>
</tbody>
</table>
unix-epoch

A DOUBLE PRECISION value that specifies some number of seconds elapsed since midnight UTC of January 1, 1970, not counting leap seconds. INTEGER values are implicitly cast to DOUBLE PRECISION.

Notes

- Millisecond (MS) and microsecond (US) values in a conversion from string to TIMESTAMP are used as part of the seconds after the decimal point. For example TO_TIMESTAMP ('12:3', 'SS:MS') is not 3 milliseconds, but 300, because the conversion counts it as 12 + 0.3 seconds. This means for the format SS:MS, the input values 12:3, 12:30, and 12:300 specify the same number of milliseconds. To get three milliseconds, use 12:003, which the conversion counts as 12 + 0.003 = 12.003 seconds.

  Here is a more complex example: TO_TIMESTAMP('15:12:02.020.001230', 'HH:MI:SS.MS.US') is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: ' \"YYYY Month\" '

- TO_TIMESTAMP, TO_TIMESTAMP_TZ, and TO_DATE skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:

  - TO_TIMESTAMP('2000 JUN', 'YYYY MON') is correct.
  - TO_TIMESTAMP('2000 JUN', 'FXYYYY MON') returns an error, because TO_TIMESTAMP expects one space only.

- The YYYY conversion from string to TIMESTAMP or DATE has a restriction if you use a year with more than four digits. You must use a non-digit character or template after YYYY, otherwise the year is always interpreted as four digits. For example, given the following arguments, TO_DATE interprets the five-digit year 20000 as a four-digit year:
=> SELECT TO_DATE('200001131', 'YYYYMMDD');
  TO_DATE
  ---------
   2000-01-13
(1 row)

Instead, use a non-digit separator after the year. For example:

=> SELECT TO_DATE('20000-1131', 'YYYY-MMDD');
  TO_DATE
  ---------
   20000-12-01
(1 row)

- In conversions from string to TIMESTAMP or DATE, the CC field is ignored if there is a YYY, YYYY or Y,YYY field. If CC is used with YY or Y, then the year is computed as (CC–1)*100+YY.

Examples

=> SELECT TO_TIMESTAMP('13 Feb 2009', 'DD Mon YYYY');
  TO_TIMESTAMP
  -------------------------
   1200-02-13 00:00:00
(1 row)
=> SELECT TO_TIMESTAMP(200120400);
  TO_TIMESTAMP
  -------------------------
   1976-05-05 01:00:00
(1 row)

See Also

Date/Time Functions

TO_TIMESTAMP_TZ

Converts a string value or a UNIX/POSIX epoch value to a TIMESTAMP WITH TIME ZONE type.

Behavior Type

Immutable if single argument form, Stable otherwise.
Syntax

`TO_TIMESTAMP_TZ ( expression, pattern )`  
`TO_TIMESTAMP ( unix-epoch )`

Parameters

<table>
<thead>
<tr>
<th><code>expression</code></th>
<th>Specifies the string value to convert, either CHAR or VARCHAR.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pattern</code></td>
<td>A CHAR or VARCHAR that specifies an output pattern string. See:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>unix-epoch</code></td>
<td>A DOUBLE PRECISION value that specifies some number of seconds elapsed since midnight UTC of January 1, 1970, not counting leap seconds. INTEGER values are implicitly cast to DOUBLE PRECISION.</td>
</tr>
</tbody>
</table>

Notes

- Millisecond (MS) and microsecond (US) values in a conversion from string to `TIMESTAMP` are used as part of the seconds after the decimal point. For example `TO_TIMESTAMP ('12:3', 'SS:MS')` is not 3 milliseconds, but 300, because the conversion counts it as 12 + 0.3 seconds. This means for the format `SS:MS`, the input values 12:3, 12:30, and 12:300 specify the same number of milliseconds. To get three milliseconds, use `12:003`, which the conversion counts as `12 + 0.003 = 12.003` seconds.

Here is a more complex example: `TO_TIMESTAMP('15:12:02.020.001230', 'HH:MI:SS.MS.US')` is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: `' \"YYYY Month\" '`

- `TO_TIMESTAMP`, `TO_TIMESTAMP_TZ`, and `TO_DATE` skip multiple blank spaces in the input string if the FX option is not used. FX must be specified as the first item in the template. For example:
- `TO_TIMESTAMP('2000 JUN', 'YYYY MON')` is correct.

- `TO_TIMESTAMP('2000 JUN', 'FXYYYY MON')` returns an error, because `TO_TIMESTAMP` expects one space only.

- The `YYYY` conversion from string to `TIMESTAMP` or `DATE` has a restriction if you use a year with more than four digits. You must use a non-digit character or template after `YYYY`, otherwise the year is always interpreted as four digits. For example, given the following arguments, `TO_DATE` interprets the five-digit year 20000 as a four-digit year:

  ```sql
  => SELECT TO_DATE('200001131', 'YYYYMMDD');
  TO_DATE
  2000-01-13
  (1 row)
  ```

  Instead, use a non-digit separator after the year. For example:

  ```sql
  => SELECT TO_DATE('20000-1131', 'YYYY-MMDD');
  TO_DATE
  20000-12-01
  (1 row)
  ```

- In conversions from string to `TIMESTAMP` or `DATE`, the `CC` field is ignored if there is a `YY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y`, then the year is computed as `(CC-1)*100+YY`.

Examples

```sql
=> SELECT TO_TIMESTAMP_TZ('13 Feb 2009', 'DD Mon YYY');
  TO_TIMESTAMP_TZ
  1200-02-13 00:00:00-05
  (1 row)
=> SELECT TO_TIMESTAMP_TZ(200120400);
  TO_TIMESTAMP_TZ
  1976-05-05 01:00:00-04
  (1 row)
```

See Also

Date/Time Functions
TO_NUMBER

Converts a string value to DOUBLE PRECISION.

Behavior Type

Stable

Syntax

TO_NUMBER ( expression, [ pattern ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Specifies the string value to convert, either CHAR or VARCHAR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A string value, either CHAR or VARCHAR, that specifies an output pattern string using one of the supported Template Patterns for Numeric Formatting. If you omit this parameter, TO_NUMBER returns a floating point.</td>
</tr>
</tbody>
</table>

Notes

To use a double quote character in the output, precede it with a double backslash. This is necessary because the backslash already has a special meaning in a string constant. For example: "\"YYYY Month\""

Note: To convert a date string to a numeric value, use the appropriate date/time function, such as EXTRACT.

Examples

```
=> SELECT TO_NUMBER('MCML', 'rn');
TO_NUMBER
-------------
```

Vertica Analytic Database (9.0.x)
It the pattern parameter is omitted, the function returns a floating point. For example:

```sql
=> SELECT TO_NUMBER('-123.456e-01');
TO_NUMBER
---------
-12.3456
```
Template Patterns for Date/Time Formatting

In an output template string (for TO_CHAR), there are certain patterns that are recognized and replaced with appropriately-formatted data from the value to be formatted. Any text that is not a template pattern is copied verbatim. Similarly, in an input template string (for anything other than TO_CHAR), template patterns identify the parts of the input data string to be looked at and the values to be found there.

Note: Vertica uses the ISO 8601:2004 style for date/time fields in Vertica *.log files. For example,
2008-09-16 14:40:59.123 TM Moveout:0x2aaaac002180 [Txn] <INFO>

Certain modifiers can be applied to any template pattern to alter its behavior as described in Template Pattern Modifiers for Date/Time Formatting.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HH</td>
<td>Hour of day (00-23)</td>
</tr>
<tr>
<td>HH12</td>
<td>Hour of day (01-12)</td>
</tr>
<tr>
<td>HH24</td>
<td>Hour of day (00-23)</td>
</tr>
<tr>
<td>MI</td>
<td>Minute (00-59)</td>
</tr>
<tr>
<td>SS</td>
<td>Second (00-59)</td>
</tr>
<tr>
<td>MS</td>
<td>Millisecond (000-999)</td>
</tr>
<tr>
<td>US</td>
<td>Microsecond (000000-999999)</td>
</tr>
<tr>
<td>SSSS</td>
<td>Seconds past midnight (0-86399)</td>
</tr>
<tr>
<td>AM or A.M. or PM or P.M.</td>
<td>Meridian indicator (uppercase)</td>
</tr>
<tr>
<td>am or a.m. or pm or p.m.</td>
<td>Meridian indicator (lowercase)</td>
</tr>
<tr>
<td>Y,YYY</td>
<td>Year (4 and more digits) with comma</td>
</tr>
<tr>
<td>YYYY</td>
<td>Year (4 and more digits)</td>
</tr>
<tr>
<td>YYYY</td>
<td>Last 3 digits of year</td>
</tr>
<tr>
<td>Pattern</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>YY</td>
<td>Last 2 digits of year</td>
</tr>
<tr>
<td>Y</td>
<td>Last digit of year</td>
</tr>
<tr>
<td>YYYY</td>
<td>ISO year (4 and more digits)</td>
</tr>
<tr>
<td>IYY</td>
<td>Last 3 digits of ISO year</td>
</tr>
<tr>
<td>IY</td>
<td>Last 2 digits of ISO year</td>
</tr>
<tr>
<td>I</td>
<td>Last digits of ISO year</td>
</tr>
<tr>
<td>BC or B.C. or AD or A.D.</td>
<td>Era indicator (uppercase)</td>
</tr>
<tr>
<td>bc or b.c. or ad or a.d.</td>
<td>Era indicator (lowercase)</td>
</tr>
<tr>
<td>MONTH</td>
<td>Full uppercase month name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>Month</td>
<td>Full mixed-case month name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>month</td>
<td>Full lowercase month name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>MON</td>
<td>Abbreviated uppercase month name (3 chars)</td>
</tr>
<tr>
<td>Mon</td>
<td>Abbreviated mixed-case month name (3 chars)</td>
</tr>
<tr>
<td>mon</td>
<td>Abbreviated lowercase month name (3 chars)</td>
</tr>
<tr>
<td>MM</td>
<td>Month number (01-12)</td>
</tr>
<tr>
<td>DAY</td>
<td>Full uppercase day name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>Day</td>
<td>Full mixed-case day name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>day</td>
<td>Full lowercase day name (blank-padded to 9 chars)</td>
</tr>
<tr>
<td>DY</td>
<td>Abbreviated uppercase day name (3 chars)</td>
</tr>
<tr>
<td>Dy</td>
<td>Abbreviated mixed-case day name (3 chars)</td>
</tr>
<tr>
<td>dy</td>
<td>Abbreviated lowercase day name (3 chars)</td>
</tr>
<tr>
<td>DDD</td>
<td>Day of year (001-366)</td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (01-31) for TIMESTAMP</td>
</tr>
<tr>
<td>Pattern</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>D</td>
<td>Day of week (1-7; Sunday is 1)</td>
</tr>
<tr>
<td>w</td>
<td>Week of month (1-5) (The first week starts on the first day of the month.)</td>
</tr>
<tr>
<td>WW</td>
<td>Week number of year (1-53) (The first week starts on the first day of the year.)</td>
</tr>
<tr>
<td>IW</td>
<td>ISO week number of year (The first Thursday of the new year is in week 1.)</td>
</tr>
<tr>
<td>CC</td>
<td>Century (2 digits)</td>
</tr>
<tr>
<td>J</td>
<td>Julian Day (days since January 1, 4712 BC)</td>
</tr>
<tr>
<td>Q</td>
<td>Quarter</td>
</tr>
<tr>
<td>RM</td>
<td>Month in Roman numerals (I-XII; I=January) (uppercase)</td>
</tr>
<tr>
<td>rm</td>
<td>Month in Roman numerals (i-xii; i=January) (lowercase)</td>
</tr>
<tr>
<td>TZ</td>
<td>Time-zone name (uppercase)</td>
</tr>
<tr>
<td>tz</td>
<td>Time-zone name (lowercase)</td>
</tr>
</tbody>
</table>

Note: For INTERVAL, DD is day of year (001-366) because day of month is undefined.

Examples

Use TO_TIMESTAMP to convert an expression using the pattern 'YYY MON':

```sql
=> SELECT TO_TIMESTAMP('2017 JUN', 'YYYY MON');
TO_TIMESTAMP
---------------------
2017-06-01 00:00:00
(1 row)
```

Use TO_DATE to convert an expression using the pattern 'YYYY-MMDD':

```sql
=> SELECT TO_DATE('2017-1231', 'YYYY-MMDD');
TO_DATE
--------
2017-12-31
```
## Template Pattern Modifiers for Date/Time Formatting

Certain modifiers can be applied to any template pattern to alter its behavior. For example, FMMonth is the Month pattern with the FM modifier.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>Time is before 12:00</td>
</tr>
<tr>
<td>AT</td>
<td>Ignored</td>
</tr>
<tr>
<td>JULIAN, JD, J</td>
<td>Next field is Julian Day</td>
</tr>
</tbody>
</table>
| FM prefix | Fill mode (suppress padding blanks and zeros)  
For example: FMMonth  
**Note**: The FM modifier suppresses leading zeros and trailing blanks that would otherwise be added to make the output of a pattern fixed width. |
| FX prefix | Fixed format global option  
For example: FX Month DD Day |
| ON       | Ignored |
| PM       | Time is on or after 12:00 |
| T        | Next field is time |
| TH suffix | Uppercase ordinal number suffix  
For example: DDTH |
| th suffix | Lowercase ordinal number suffix  
For example: DDth |
| TM prefix | Translation mode (print localized day and month names based on lc_messages). For example: TMMonth |
## Template Patterns for Numeric Formatting

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Value with the specified number of digits</td>
</tr>
<tr>
<td>0</td>
<td>Value with leading zeros</td>
</tr>
<tr>
<td>. (period)</td>
<td>Decimal point</td>
</tr>
<tr>
<td>, (comma)</td>
<td>Group (thousand) separator</td>
</tr>
<tr>
<td>PR</td>
<td>Negative value in angle brackets</td>
</tr>
<tr>
<td>S</td>
<td>Sign anchored to number (uses locale)</td>
</tr>
<tr>
<td>L</td>
<td>Currency symbol (uses locale)</td>
</tr>
<tr>
<td>D</td>
<td>Decimal point (uses locale)</td>
</tr>
<tr>
<td>G</td>
<td>Group separator (uses locale)</td>
</tr>
<tr>
<td>MI</td>
<td>Minus sign in specified position (if number &lt; 0)</td>
</tr>
<tr>
<td>PL</td>
<td>Plus sign in specified position (if number &gt; 0)</td>
</tr>
<tr>
<td>SG</td>
<td>Plus/minus sign in specified position</td>
</tr>
<tr>
<td>RN</td>
<td>Roman numeral (input between 1 and 3999)</td>
</tr>
<tr>
<td>TH or 'th'</td>
<td>Ordinal number suffix</td>
</tr>
<tr>
<td>V</td>
<td>Shift specified number of digits (see notes)</td>
</tr>
<tr>
<td>EEEE</td>
<td>Scientific notation (not implemented yet)</td>
</tr>
</tbody>
</table>
Usage

- A sign formatted using SG, PL, or MI is not anchored to the number; for example:
  - `TO_CHAR(-12, 'S9999')` produces `-12'
  - `TO_CHAR(-12, 'MI9999')` produces `-12'

- 9 results in a value with the same number of digits as there are 9s. If a digit is not available it outputs a space.

- TH does not convert values less than zero and does not convert fractional numbers.

- V effectively multiplies the input values by $10^n$, where $n$ is the number of digits following V. TO_CHAR does not support the use of V combined with a decimal point. For example: 99.9V99 is not allowed.

Geospatial Functions

The following topics describe the Vertica geospatial functions.

Function-Naming Conventions

The geospatial functions use the following naming conventions:

- The `ST_<function_name>` functions are compliant with the latest Open Geospatial Consortium standard OGC SFA-SQL version 1.2.1 (reference. number is OGC 06-104r4, date: 2010-08-04). Currently, some `ST_<function_name>` functions may not support all data types. Each function page contains details about the supported data types.

  Note: Some functions, such as `ST_GeomFromText`, are based on previous versions of the standard.

- The `STV_<function_name>` functions are unique to Vertica and not compliant with OGC standards. Each function page explains its functionality in detail.
Verifying Spatial Objects Validity

Many spatial functions do not verify the validity of the parameters. If you pass an invalid spatial object to an ST_ or STV_ function, the function may return an error or produce incorrect results.

To avoid this issue, OpenText recommends that you first run ST_IsValid on all spatial objects to verify their validity. If your object is not valid, run STV_IsValidReason to get information about the location of the invalidity.

Note: If you pass a valid polygon to STV_IsValidReason, it returns NULL.

ST_AsText

Creates the Well-Known Text (WKT) representation of a spatial object. Use this function when you need to specify a spatial object in ASCII form.

The Open Geospatial Consortium (OGC) defines the format of a WKT string in the Simple Feature Access Part 1 - Common Architecture specification.

Behavior Type

Immutable

Syntax

ST_AsText(  g  )

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Spatial object for which you want the WKT string, type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

Returns

LONG VARCHAR
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use ST_AsText.

Retrieve WKB and WKT representations:

```sql
=> CREATE TABLE locations (id INTEGER, name VARCHAR(100), geom1 GEOMETRY(800), geom2 GEOGRAPHY);
=> CREATE TABLE
=> COPY locations
  (id, geom1x FILLER LONG VARCHAR(800), geom1 AS ST_GeomFromText(geom1x), geom2x FILLER LONG VARCHAR(800),
   geom2 AS ST_GeographyFromText(geom2x))
FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> 1|POINT(2 3)|
=> 2|LINESTRING(2 4,1 5)|
=> 3|POLYGON((-70.96 43.27,-70.67 42.95,-66.90 44.74,-67.81 46.08,-67.81 47.20,-69.22 47.43,-71.09 45.25,-70.96 43.27))
=> .
=> SELECT id, ST_AsText(geom1),ST_AsText(geom2) FROM locations ORDER BY id ASC;
  id | ST_AsText                                           | ST_AsText                                           |
  1 | POINT (2 3)                                         |                                                   |
  2 | LINESTRING (2 4, 1 5)                               |                                                   |
  3 | POLYGON ((-70.96 43.27, -70.67 42.95, -66.9 44.74, -67.81 46.08, -67.81 47.2, -69.22 47.43, -71.09 45.25, -70.96 43.27)) |
Calculate the length of a WKT using the Vertica SQL function **LENGTH**:

```sql
=> SELECT LENGTH(ST_AsText(St_GeomFromText('POLYGON ((-1 2, 0 3, 1 2, 0 1, -1 2))')));
```

```
LENGTH
--------
 37
(1 row)
```

**See Also**

- **ST_AsBinary**

**ST_Area**

Calculates the area of a spatial object.

The units are:

- GEOMETRY objects: spatial reference system identifier (SRID) units
- GEOGRAPHY objects: square meters

**Behavior Type**

Immutable

**Syntax**

```
ST_Area( g )
```

**Arguments**

| g       | Spatial object for which you want to calculate the area, type GEOMETRY or GEOGRAPHY |
Returns

FLOAT

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_Area.

Calculate the area of a polygon:

```sql
=> SELECT ST_Area(ST_GeomFromText('POLYGON((0 0, 1 0, 1, 0, 0, 0))'));
ST_Area
--------
  1
(1 row)
```

Calculate the area of a multipolygon:

```sql
=> SELECT ST_Area(ST_GeomFromText('MultiPolygon(((0 0, 1 0, 1, 0, 0)),
    ((2, 2, 3, 4, 6, 3, 2, 2)))'));
ST_Area
--------
  3
(1 row)
```

Suppose the polygon has a hole, as in the following figure.
Calculate the area, excluding the area of the hole:

```sql
=> SELECT ST_Area(ST_GeomFromText('POLYGON((2 2,5 8,2 2),
             (4 3,5 4,6 3,4 3))'));
  ST_Area
  -------
    8
(1 row)
```

Calculate the area of a geometry collection:

```sql
=> SELECT ST_Area(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((20.5 20.45,
             20.51 20.52,20.69 20.32,20.5 20.45)),
             POLYGON((10 20,30 40,25 50,10 20)))));
  ST_Area
  -------
  150.0073
(1 row)
```

Calculate the area of a geography object:

```sql
=> SELECT ST_Area(ST_GeographyFromText('POLYGON((20.5 20.45,20.51 20.52,
             20.69 20.32,20.5 20.45))'));
  ST_Area
  -------
84627437.116037
(1 row)
```

**ST_AsBinary**

Creates the Well-Known Binary (WKB) representation of a spatial object. Use this function when you need to convert an object to binary form for porting spatial data to or from other applications.

The Open Geospatial Consortium (OGC) defines the format of a WKB representation in the Simple Feature Access Part 1 - Common Architecture specification.
Behavior Type

Immutable

Syntax

```
ST_AsBinary( g )
```

Arguments

| g | Spatial object for which you want the WKB, type GEOMETRY or GEOGRAPHY |

Returns

LONG VARBINARY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Example

The following example shows how to use `ST_AsBinary`.

Retrieve WKB and WKT representations:

```sql
=> CREATE TABLE locations (id INTEGER, name VARCHAR(100), geom1 GEOMETRY(800), geom2 GEOGRAPHY);
CREATE TABLE
=> COPY locations
   (id, geom1 FILLER LONG VARCHAR(800), name AS ST_GeomFromText(geom1), geom2 FILLER LONG VARCHAR(800), geom2 AS ST_GeographyFromText(geom2))
FROM stdin;
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.

1|POINT(2 3)|
2|LINESTRING(2 4, 1 5)|
3|POLYGON((-70.96 43.27, -70.67 42.95, -66.90 44.74, -67.81 46.08, -67.81 47.20, -69.22 47.43, -71.09 45.25, -70.96 43.27))
=> SELECT id, ST_AsText(geom1), ST_AsText(geom2) FROM locations ORDER BY id ASC;

<table>
<thead>
<tr>
<th>id</th>
<th>ST_AsText</th>
<th>ST_AsText</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (2 3)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LINESTRING (2 4, 1 5)</td>
<td>POLYGON ((-70.96 43.27, -70.67 42.95, -66.90 44.74, -67.81 46.08, -67.81 47.20, -69.22 47.43, -71.09 45.25, -70.96 43.27))</td>
</tr>
</tbody>
</table>

Calculate the length of a WKB using the Vertica SQL function `LENGTH`:

```sql
=> SELECT LENGTH(ST_AsBinary(St_GeomFromText('POLYGON ((-1 2, 0 3, 1 2, 0 1, -1 2))')));

LENGTH
-----
93
```

See Also

`ST_AsText`
ST_Boundary

Calculates the boundary of the specified GEOMETRY object. An object's boundary is the set of points that define the limit of the object.

For a linestring, the boundary is the start and end points. For a polygon, the boundary is a linestring that begins and ends at the same point.

Behavior Type

Immutable

Syntax

\[
\text{ST\_Boundary}( \text{ } g \text{ })
\]

Arguments

\(
g\)
Spatial object for which you want the boundary, type GEOMETRY

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Boundary.

Returns a linestring that represents the boundary:

```sql
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((-1 -1, 2, 0, 1, -1 -1))')));
ST_AsText
-----------
LINESTRING(-1 -1, 2 2, 0 1, -1 -1)
(1 row)
```

Returns a multiline string that contains the boundaries of both polygons:

```sql
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((2 2, 5 5, 8 2, 2 2),
(4 3, 5 4, 6 3, 4 3))')));
ST_AsText
--------------------------
MULTILINESTRING ((2 2, 5 5, 8 2, 2 2), (4 3, 5 4, 6 3, 4 3))
(1 row)
```

The boundary of a linestring is its start and end points:

```sql
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(1 1, 2 3, 4 4)')));
ST_AsText
----------
MULTIPOINT (1 1, 4 4)
(1 row)
```

A closed linestring has no boundary because it has no start and end points:

```sql
=> SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(1 1, 2 3, 4 1 1)')));
ST_AsText
----------
MULTIPOINT EMPTY
(1 row)
```
ST_Buffer

Creates a GEOMETRY object greater than or equal to a specified distance from the boundary of a spatial object. The distance is measured in Cartesian coordinate units. ST_Buffer does not accept a distance size greater than +1e15 or less than –1e15.

Behavior Type

Immutable

Syntax

ST_Buffer( g, d )

Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Spatial object for which you want to calculate the buffer, type GEOMETRY</td>
</tr>
<tr>
<td>d</td>
<td>Distance from the object in Cartesian coordinate units, type FLOAT</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Usage Tips

- If you specify a positive distance, ST_Buffer returns a polygon that represents the points within or equal to the distance outside the object. If you specify a negative distance, ST_Buffer returns a polygon that represents the points within or equal to the distance inside the object.

- For points, multipoints, linestrings, and multilinestrings, if you specify a negative distance, ST_Buffer returns an empty polygon.

- The Vertica Place version of ST_Buffer returns the buffer as a polygon, so the buffer object has corners at its vertices. It does not contain rounded corners.

### Example

The following example shows how to use ST_Buffer.

**Returns a GEOMETRY object:**

```sql
=> SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POLYGON((0 1,4,4,3,0 1)'),1));

ST_AsText
---------------------------
POLYGON ((-0.188847498856 -0.159920845081, -1.12155598386, 0.649012935089, 0.290814745534 4.76344136152, 0.814758063466 5.02541302048, 4.95372324225 3.68665254814, 5.04124517538 2.45512549204, -0.188847498856 -0.159920845081))
(1 row)
```

### ST_Centroid

Calculates the geometric center—the centroid—of a spatial object. If points or linestrings or both are present in a geometry with polygons, only the polygons contribute to the calculation of the centroid. Similarly, if points are present with linestrings, the points do not contribute to the calculation of the centroid.

To calculate the centroid of a GEOGRAPHY object, see the examples for STV_Geometry and STV_Geography.
Behavior Type

Immutable

Syntax

\texttt{ST\_Centroid( g )}

Arguments

| \( g \) | Spatial object for which you want to calculate the centroid, type \texttt{GEOMETRY} |

Returns

\texttt{GEOMETRY (POINT only)}

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Centroid.

Calculate the centroid for a polygon:

```sql
=> SELECT ST_AsText(ST_Centroid(STGeomFromText('POLYGON((-1 -1, 2, -1 2, -1 -1))')));
  ST_AsText
---------------------
  POINT (-0 1)
(1 row)
```

Calculate the centroid for a multipolygon:

```sql
=> SELECT ST_AsText(ST_Centroid(STGeomFromText('MULTIPOLYGON(((1 0, 2 1, 0, 1 0)),((-1 -1, 2, -1 2, -1 -1)))));
  ST_AsText
---------------------
  POINT (0.166666666667 0.933333333333)
(1 row)
```

This figure shows the centroid for the multipolygon.

ST_Contains

Determines if a spatial object is entirely inside another spatial object without existing only on its boundary. Both arguments must be the same spatial data type. Either specify two GEOMETRY objects or two GEOGRAPHY objects.
If an object such as a point or linestring only exists along a spatial object’s boundary, then ST_Contains returns false. The interior of a linestring is all the points on the linestring except the start and end points.

ST_Contains(g1, g2) is functionally equivalent to ST_Within(g2, g1).

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type
Immutable

Syntax

ST_Contains( g1, g2 )

[USING PARAMETERS spheroid={true | false}] )

Arguments

<table>
<thead>
<tr>
<th>g1</th>
<th>Spatial object, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2</td>
<td>Spatial object, type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

Parameters

| spheroid = {true | false} | (Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. |
|--------------------------|-------------------------------------------------|
|                          | Default: False                                  |

Returns

BOOLEAN
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatible GEOGRAPHY pairs:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-Point</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linestring-Point</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Polygon-Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon-Point</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_Contains.

The first polygon does not completely contain the second polygon:

```sql
=> SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 0 -1,0 2))'),
                      ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
ST_Contains
----------
 1
(1 row)
```
If a point is on a linestring, but not on an end point:

```sql
=> SELECT ST_Contains(ST_GeomFromText('LINESTRING(20 20,30 30)'),
                      ST_GeomFromText('POINT(25 25)'));
ST_Contains
-----------
t
(1 row)
```

If a point is on the boundary of a polygon:

```sql
=> SELECT ST_Contains(ST_GeographyFromText('POLYGON((20 20,30 30,30 25,25 20))'),
                      ST_GeographyFromText('POINT(20 20)'));
ST_Contains
-----------
f
(1 row)
```
Two spatially equivalent polygons:

```sql
=> SELECT ST_Contains(ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),
                      ST_GeomFromText('POLYGON((0 3, -1 2, 0 1, 0 3))));
  ST_Contains
-------------------
  t
(1 row)
```

See Also

- **ST_Overlaps**
- **ST_Within**

**ST_ConvexHull**

Calculates the smallest convex GEOMETRY object that contains a GEOMETRY object.
Behavior Type

Immutable

Syntax

ST_ConvexHull( g )

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Spatial object for which you want the convex hull, type GEOMETRY</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_ConvexHull.
For a pair of points in a geometry collection:

```sql
=> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText('GEOMETRYCOLLECTION(POINT(1 1), POINT(0 0)))));
ST_AsText
---------------
LINESTRING (1 1, 0 0)
(1 row)
```

For a geometry collection:

```sql
=> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText('GEOMETRYCOLLECTION(LINESTRING(2.5 3, -2 1.5), POLYGON((0 1, 1 3, 1 -2, 0 1)))')));
ST_AsText
---------------
POLYGON ((1 -2, -2 1.5, 1 3, 2.5 3, 1 -2))
(1 row)
```

The solid lines represent the original geometry collection and the dashed lines represent the convex hull.

**ST_Crosses**

Determines if one GEOMETRY object spatially crosses another GEOMETRY object. If two objects touch only at a border, ST_Crosses returns FALSE.

Two objects spatially cross when both of the following are true:

- The two objects have some, but not all, interior points in common.
- The dimension of the result of their intersection is less than the maximum dimension of the two objects.
Behavior Type
Immutable

Syntax
`ST_Crosses( g1, g2 )`

Arguments
<table>
<thead>
<tr>
<th>$g1$</th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g2$</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>

Returns
BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Crosses.

```sql
=> SELECT ST_Crosses(ST_GeomFromText('LINESTRING(-1 3,1 4)'),
    ST_GeomFromText('LINESTRING(-1 4,1 3)'));
ST_Crosses
----------
t
(1 row)

=> SELECT ST_Crosses(ST_GeomFromText('LINESTRING(-1 1,1 2)'),
    ST_GeomFromText('POLYGON((1 1,0 -1,3 -1,2 1,1 1))'));
ST_Crosses
----------
f
(1 row)
```
ST_Crosses

- FromText

=\>	SELECT ST_Crosses(ST_GeomFromText('POINT(-1 4)'),
                      ST_GeomFromText('LINESTRING(-1 4,1 3)'));

ST_Crosses

---

1 row

ST_Difference

Calculates the part of a spatial object that does not intersect with another spatial object.

Behavior Type

Immutable

Syntax

ST_Difference( g1, g2 )

Arguments

| \( g1 \) | Spatial object, type GEOMETRY |
| \( g2 \) | Spatial object, type GEOMETRY |

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Difference.

Two overlapping linestrings:

```sql
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('LINESTRING(0 0,0 2)'),
    ST_GeomFromText('LINESTRING(0 1,0 2)')));
ST_AsText
------------------------
LINESTRING (0 0, 0 1)
(1 row)

=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('LINESTRING(0 0,0 3)'),
    ST_GeomFromText('LINESTRING(0 1,0 2)')));
ST_AsText
------------------------
MULTILINESTRING ((0 0, 0 1), (0 2, 0 3))
(1 row)
```

Two overlapping polygons:
Two non-intersecting polygons:

```sql
=> SELECT ST_AsText(ST_Difference(ST_GeomFromText('POLYGON((0 1,0 3,2 3,2 1,0 1))'),
                                  ST_GeomFromText('POLYGON((0 0,0 2,2 2,0 0)'))));
-----------------------------
POLYGON ((0 2, 0 3, 2 3, 2 2, 0 2))
(1 row)
```

**ST_Disjoint**

Determines if two GEOMETRY objects do not intersect or touch.

If **ST_Disjoint** returns TRUE for a pair of GEOMETRY objects, **ST_Intersects** returns FALSE for the same two objects.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

**Behavior Type**

Immutable

**Syntax**

```sql
ST_Disjoint( g1, g2 )
            [USING PARAMETERS spheroid={true | false} ]
```

**Arguments**

<table>
<thead>
<tr>
<th></th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>g1</td>
<td></td>
</tr>
<tr>
<td>g2</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

```
spheroid = {true | false}  (Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84.
                           Default: False
```

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatible GEOGRAPHY pairs:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-Point</td>
<td>No</td>
</tr>
<tr>
<td>Linestring-Point</td>
<td>No</td>
</tr>
<tr>
<td>Polygon-Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon-Point</td>
<td>No</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Disjoint.

Two non-intersecting or touching polygons:

```sql
=> SELECT ST_Disjoint (ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
  ST_GeomFromText('POLYGON((1 0, 1 1, 2 2, 1 0))'));
```

```
t
(1 row)
```

Two intersecting linestrings:

```sql
=> SELECT ST_Disjoint(ST_GeomFromText('LINESTRING(-1 2,0 3)'),
  ST_GeomFromText('LINESTRING(0 2,-1 3)'));
```

```
f
(1 row)
```

Two polygons touching at a single point:

```sql
=> SELECT ST_Disjoint (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),
  ST_GeomFromText('POLYGON((0 2, 1 1, 1 2, 0 2))'));
```

```
f
(1 row)
```

See Also

- ST_Intersects

ST_Distance

Calculates the shortest distance between two spatial objects. For GEOMETRY objects, the
distance is measured in Cartesian coordinate units. For GEOGRAPHY objects, the distance is
measured in meters.

Parameters $g1$ and $g2$ must be both GEOMETRY objects or both GEOGRAPHY objects.
Behavior Type

Immutable

Syntax

ST_Distance( g1, g2
        [USING PARAMETERS spheroid={ true | false } ] )

Arguments

<table>
<thead>
<tr>
<th>g1</th>
<th>Spatial object, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2</td>
<td>Spatial object, type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

Parameters

| spheroid = { true | false } | (Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84. Default: False |

Returns

FLOAT

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Multilinestring
<table>
<thead>
<tr>
<th>Compatible GEOGRAPHY pairs:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Type</strong></td>
</tr>
<tr>
<td>Point-Point</td>
</tr>
<tr>
<td>Linestring-Point</td>
</tr>
<tr>
<td>Multilinestring-Point</td>
</tr>
<tr>
<td>Polygon-Point</td>
</tr>
<tr>
<td>Multipoint-Point</td>
</tr>
<tr>
<td>Multipoint-Multilinestring</td>
</tr>
<tr>
<td>Multipolygon-Point</td>
</tr>
</tbody>
</table>

**Recommendations**

Vertica recommends pruning invalid data before using `ST_Distance`. Invalid geography values could return non-guaranteed results.

**Examples**

The following examples show how to use `ST_Distance`.

Distance between two polygons:
SELECT ST_Distance(ST_GeomFromText('POLYGON((-1 -1,2,0 1,-1 -1))'),
                        ST_GeomFromText('POLYGON((5 2,7 4,5 5,2))'));

3

(1 row)

Distance between a point and a linestring in meters:

=> SELECT ST_Distance(ST_GeographyFromText('POINT(31.75 31.25)'),'LINESTRING(32 32,32 35,35,32 35,32 32)'));

86690.3950562969

(1 row)

### ST_Envelope

Calculates the minimum bounding rectangle that contains the specified GEOMETRY object.

#### Behavior Type

Immutable

#### Syntax

ST_Envelope( g )

#### Arguments

| g    | Spatial object for which you want to find the minimum bounding rectangle, type GEOMETRY |

#### Returns

GEOMETRY
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use ST_Envelope.

Returns the minimum bounding rectangle:

```sql
=> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0))')));

POLYGON ((0 -1.5, 3 -1.5, 3 2, 0 2, 0 -1.5))
(1 row)
```
ST_Equals

Determines if two spatial objects are spatially equivalent. The coordinates of the two objects and their WKT/WKB representations must match exactly for ST_Equals to return TRUE.

The order of the points do not matter in determining spatial equivalence:

- LINESTRING(1 2, 4 3) equals LINESTRING(4 3, 1 2).
- POLYGON ((0 0, 1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0)) equals POLYGON((1 1, 1 2, 2 2, 2 1, 3 0, 1.5 -1.5, 0 0, 1 1)).
- MULTILINESTRING((1 2, 4 3),(0 0, -1 -4)) equals MULTILINESTRING((0 0, -1 -4),(1 2, 4 3)).

Coordinates are stored as FLOAT types. Thus, rounding errors are expected when importing Well-Known Text (WKT) values because the limitations of floating-point number representation.

$g1$ and $g2$ must both be GEOMETRY objects or both be GEOGRAPHY objects. Also, $g1$ and $g2$ cannot both be of type GeometryCollection.

Behavior Type

Immutable

Syntax

`ST_Equals( g1, g2 )`

Arguments

<table>
<thead>
<tr>
<th>$g1$</th>
<th>Spatial object to compare to $g2$, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g2$</td>
<td>Spatial object to compare to $g1$, type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

Returns

BOOLEAN
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_Equals.

Two linestrings:

```sql
=> SELECT ST_Equals (ST_GeomFromText('LINESTRING(-1 2, 0 3)'),
                        ST_GeomFromText('LINESTRING(0 3, -1 2)'));
ST_Equals
----------
t
(1 row)
```

Two polygons:

```sql
=> SELECT ST_Equals (ST_GeographyFromText('POLYGON((43.22 42.21, 40.3 39.88, 42.1 50.03, 43.22 42.21))'),
                        ST_GeographyFromText('POLYGON((43.22 42.21, 40.3 39.88, 42.1 50.31, 43.22 42.21))'));
ST_Equals
----------	f
(1 row)
```
**ST_GeographyFromText**

Converts a Well-Known Text (WKT) string into its corresponding GEOGRAPHY object. Use this function to convert a WKT string into the format expected by the Vertica Place functions.

A GEOGRAPHY object is a spatial object with coordinates (longitude, latitude) defined on the surface of the earth. Coordinates are expressed in degrees (longitude, latitude) from reference planes dividing the earth.

The maximum size of a GEOGRAPHY object is 10 MB. If you pass a WKT to ST_GeographyFromText, the result is a spatial object whose size is greater than 10 MB, ST_GeographyFromText returns an error.


**Behavior Type**

Immutable

**Syntax**

```
ST_GeographyFromText( wkt [ USING PARAMETERS ignore_errors='y'|'n' ] )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wkt</code></td>
<td>Well-Known Text (WKT) string of a GEOGRAPHY object, type LONG VARCHAR</td>
</tr>
<tr>
<td><code>ignore_errors</code></td>
<td>(Optional) ST_GeographyFromText returns the following, based on the parameters supplied:</td>
</tr>
<tr>
<td></td>
<td>NULL—If <code>wkt</code> is invalid and <code>ignore_errors='y'</code>.</td>
</tr>
<tr>
<td></td>
<td>Error—If <code>wkt</code> is invalid and <code>ignore_errors='n'</code> or is unspecified.</td>
</tr>
</tbody>
</table>

**Returns**

GEOGRAPHY
**Supported Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use `ST_GeographyFromText`.

Convert WKT into a GEOGRAPHY object:

```sql
=> CREATE TABLE wkt_ex (g GEOGRAPHY);
CREATE TABLE
=> INSERT INTO wkt_ex VALUES(ST_GeographyFromText('POLYGON((1 2,3 4,2 3,1 2))'));
OUTPUT
--------
1
(1 row)
```

**ST_GeographyFromWKB**

Converts a Well-Known Binary (WKB) value into its corresponding GEOGRAPHY object. Use this function to convert a WKB into the format expected by Vertica Place functions.

A GEOGRAPHY object is a spatial object defined on the surface of the earth. Coordinates are expressed in degrees (longitude, latitude) from reference planes dividing the earth. All calculations are in meters.
The maximum size of a GEOGRAPHY object is 10 MB. If you pass a WKB to ST_GeographyFromWKB that results in a spatial object whose size is greater than 10 MB, ST_GeographyFromWKB returns an error.

The Open Geospatial Consortium (OGC) defines the format of a WKB representation in Section 8 in the Simple Feature Access Part 1 - Common Architecture specification.

Behavior Type

Immutable

Syntax

ST_GeographyFromWKB( wkb [ USING PARAMETERS ignore_errors='y'|'n' ] )

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wkb</td>
<td>Well-Known Binary (WKB) value of a GEOGRAPHY object, type LONG VARBINARY</td>
</tr>
<tr>
<td>ignore_errors</td>
<td>(Optional) ST_GeographyFromWKB returns the following, based on the parameters supplied:</td>
</tr>
<tr>
<td></td>
<td>• NULL—If wkb is invalid and ignore_errors='y'.</td>
</tr>
<tr>
<td></td>
<td>• Error—If wkb is invalid and ignore_errors='n' or is unspecified.</td>
</tr>
</tbody>
</table>

Returns

GEOGRAPHY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Geometry Type</td>
<td>Vertica</td>
<td>SQL Server</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use `ST_GeographyFromWKB`.

Convert WKB into a GEOGRAPHY object:

```
=> CREATE TABLE wkb_ex (g GEOGRAPHY);
CREATE TABLE
=> INSERT INTO wkb_ex VALUES(ST_GeographyFromWKB(X'01030000000100000001000000 ... ));
OUTPUT
--------
   1
(1 row)
```

**ST_GeoHash**

Returns a GeoHash in the shape of the specified geometry.

**Behavior Type**

Immutable

**Syntax**

```
ST_GeoHash( SpatialObject [ USING PARAMETERS numchars=n ] )
```

**Arguments**

| `Spatial object` | A GEOMETRY or GEOGRAPHY spatial object. Inputs must be in polar coordinates (-180 <= x <= 180 and -90 <= y <= 90) for all points inside the given geometry. |
$n$ Specifies the length, in characters, of the returned GeoHash.

## Returns

GEOHASH

### Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

## Examples

The following examples show how to use ST_PointFromGeoHash.

**Generate a full precision GeoHash for the specified geometry:**

```sql
=> SELECT ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)'));
ST_GeoHash
---------------------------
kpf0rkn3zmcswks75010
(1 row)
```

**Generate a GeoHash based on the first five characters of the specified geometry:**

```sql
=> select ST_GeoHash(ST_GeographyFromText('POINT(3.14 -1.34)')USING PARAMETERS numchars=5);
ST_GeoHash
----------
kpf0r
```
ST_GeometryN

Returns the $n^{th}$ geometry within a geometry object.

If $n$ is out of range of the index, then NULL is returned.

Behavior Type

Immutable

Syntax

$$\text{ST_GeometryN}( g , n )$$

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>Spatial object of type GEOMETRY.</td>
</tr>
<tr>
<td>$n$</td>
<td>The geometry's index number, 1-based.</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Linestring
- Yes
- Yes
- Yes

### Multilinestring
- Yes
- Yes
- Yes

### Polygon
- Yes
- Yes
- Yes

### Multipolygon
- Yes
- Yes
- Yes

### GeometryCollection
- No
- No
- No

## Examples

The following examples show how to use ST_GeometryN.

**Return the second geometry in a multipolygon:**

```sql
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
=> SELECT gid, ST_AsText(ST_GeometryN(geom, 2)) FROM multipolygon_geom;
gid | ST_AsText
---------------------
 9 | POLYGON ((0 0, 0 5, 1 0, 0 0))
(1 row)
```

**Return all the geometries within a multipolygon:**

```sql
=> CREATE TABLE multipolygon_geom (gid int, geom GEOMETRY(1000));
CREATE TABLE
=> COPY multipolygon_geom(gid, gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin
delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> MULTIPOLYGON(((2 6, 2 9, 6 9, 7 7, 4 6, 2 6)),((0 0, 0 5, 1 0, 0 0)),((0 2, 2 5, 4 5, 0 2)))
=> CREATE TABLE series_numbers (numbs int);
CREATE TABLE
=> COPY series_numbers FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

>> 1
>> 2
>> 3
>> 4
>> 5
>> \. 
=> SELECT numbs, ST_AsText(ST_GeometryN(geom, numbs))
FROM multipolygon_geom, series_numbers
```
WHERE ST_AsText(ST_GeometryN(geom, nums)) IS NOT NULL
ORDER BY nums ASC;

<table>
<thead>
<tr>
<th>nums</th>
<th>ST_AsText</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POLYGON ((2 6, 2 9, 6 9, 7 7, 4 6, 2 6))</td>
</tr>
<tr>
<td>2</td>
<td>POLYGON ((0 0, 0 5, 1 0, 0 0))</td>
</tr>
<tr>
<td>3</td>
<td>POLYGON ((0 2, 2 5, 4 5, 0 2))</td>
</tr>
</tbody>
</table>
(3 rows)

See Also

ST_NumGeometries

ST_GeometryType

Determines the class of a spatial object.

Behavior Type

Immutable

Syntax

ST_GeometryType( g )

Arguments

| g   | Spatial object for which you want the class, type GEOMETRY or GEOGRAPHY |

Returns

VARCHAR

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry Type</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use `ST_GeometryType`.

Returns spatial class:

```sql
=> SELECT ST_GeometryType(ST_GeomFromText('GEOMETRYCOLLECTION(LINESTRING(1 1, 2 2), POLYGON((1 3,4 5,2,1 3)))'));

ST_GeometryType
-------------------
ST_GeometryCollection  
(1 row)
```

**ST_GeomFromGeoHash**

Returns a polygon in the shape of the specified GeoHash.

**Behavior Type**

Immutable

**Syntax**

```
ST_GeomFromGeoHash(GeoHash)
```
Arguments

| GeoHash | A valid GeoHash string of arbitrary length. |

Returns

GEOGRAPHY

Examples

The following examples show how to use ST_GeomFromGeoHash.

Converts a GeoHash string to a Geography object and back to a GeoHash

```sql
=> SELECT ST_Geom(ST_GeomFromGeoHash('vert1c9'));
ST_Geom
----------------------------------------
vert1c9
(1 row)
```

Returns a polygon of the specified GeoHash and uses ST_AsText to convert the polygon, rectangle map tile, into Well-Known Text:

```sql
=> SELECT ST_AsText(ST_GeomFromGeoHash('drt3jj9n4dpcbcdef'));
ST_AsText
----------------------------------------
POLYGON ((-71.1459699298 42.3945346513, -71.1459699297 42.3945346513, -71.1459699297 42.3945346513, -71.1459699298 42.3945346513, -71.1459699298 42.3945346513))
(1 row)
```

Returns multiple polygons and their areas for the specified GeoHashes. The polygon for the high level GeoHash (1234) has a significant area, while the low level GeoHash (1234567890bcdefhjkmn) has an area of zero.

```sql
=> SELECT ST_Area(short) short_area, ST_AsText(short) short_WKT, ST_Area(long) long_area, ST_AsText (long) long_WKT from (SELECT ST_GeomFromGeoHash('1234') short, ST_GeomFromGeoHash ('1234567890bcdefhjkmn') long) as foo;
-=[ RECORD 1 ]-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
| short_area | 24609762.8991076 |
| short_WKT  | POLYGON ((-122.34375 -88.2421875, -121.9921875 -88.2421875, -121.9921875 -88.06640625, -122.34375 -88.06640625, -122.34375 -88.2421875)) |
| long_area  | 0 |
| long_WKT   | POLYGON ((-122.196077187 -88.2297377551, -122.196077187 -88.2297377551, -122.196077187 -88.2297377551, -122.196077187 -88.2297377551)) |
```

(True不知所措。)
**ST_GeomFromText**

Converts a Well-Known Text (WKT) string into its corresponding GEOMETRY object. Use this function to convert a WKT string into the format expected by the Vertica Place functions.

A GEOMETRY object is a spatial object defined by the coordinates of a plane. Coordinates are expressed as points on a Cartesian plane \((x, y)\). SRID values of 0 to \(2^{32}-1\) are valid. SRID values outside of this range will generate an error.

The maximum size of a GEOMETRY object is 10 MB. If you pass a WKT to ST_GeomFromText and the result is a spatial object whose size is greater than 10 MB, ST_GeomFromText returns an error.

The Open Geospatial Consortium (OGC) defines the format of a WKT representation. See section 7 in the Simple Feature Access Part 1 - Common Architecture specification.

**Behavior Type**

Immutable

**Syntax**

```
ST_GeomFromText( wkt [, sridd] [ USING PARAMETERS ignore_errors=('y'='n') ]
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wkt</strong></td>
<td>Well-Known Text (WKT) string of a GEOMETRY object, type LONG VARCHAR.</td>
</tr>
<tr>
<td><strong>sridd</strong></td>
<td>(Optional when not performing operations) Spatial reference system identifier (SRID) of the GEOMETRY object, type INTEGER. The SRID is stored in the GEOMETRY object, but does not influence the results of spatial computations.</td>
</tr>
<tr>
<td><strong>ignore_errors</strong></td>
<td>(Optional) ST_GeomFromText returns the following, based on parameters supplied:</td>
</tr>
</tbody>
</table>
If \textit{wkt} is invalid and \textit{ignore\_errors}='y'.

- Error—If \textit{wkt} is invalid and \textit{ignore\_errors}='n' or is unspecified.

Returns

\textbf{GEOMETRY}

\section*{Supported Data Types}

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
</tr>
</tbody>
</table>

\section*{Example}

The following example shows how to use \texttt{ST\_GeomFromText}.

\textbf{Convert WKT into a GEOMETRY object:}

```sql
=> SELECT ST_Area(ST_GeomFromText('POLYGON((1 1,2 3,3 5,0 5,1 -2,0 0,1 1))'));
  ST_Area
  --------
   6
(1 row)
```
ST_GeomFromWKB

Converts the Well-Known Binary (WKB) value to its corresponding GEOMETRY object. Use this function to convert a WKB into the format expected by many of the Vertica Place functions.

A GEOMETRY object is a spatial object with coordinates \((x,y)\) defined in the Cartesian plane.

The maximum size of a GEOMETRY object is 10 MB. If you pass a WKB to ST_GeomFromWKB and the result is a spatial object whose size is greater than 10 MB, ST_GeomFromWKB returns an error.

The Open Geospatial Consortium (OGC) defines the format of a WKB representation in section 8 in the Simple Feature Access Part 1 - Common Architecture specification.

Behavior Type

Immutable

Syntax

\[
\text{ST\_GeomFromWKB}(\ wkb[, \ sr\dd] \ [ \text{USING PARAMETERS} \ ignore\_errors=\{'y'|'n'\} \ ])
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( wkb )</td>
<td>Well-Known Binary (WKB) value of a GEOMETRY object, type LONG VARBINARY</td>
</tr>
<tr>
<td>( sr\dd )</td>
<td>(Optional) Spatial reference system identifier (SRID) of the GEOMETRY object, type INTEGER. The SRID is stored in the GEOMETRY object, but does not influence the results of spatial computations.</td>
</tr>
<tr>
<td>( ignore_errors )</td>
<td>(Optional) ST_GeomFromWKB returns the following, based on the parameters supplied:</td>
</tr>
<tr>
<td></td>
<td>- NULL—If ( wkb ) is invalid and ( ignore_errors='y' ).</td>
</tr>
<tr>
<td></td>
<td>- Error—If ( wkb ) is invalid and ( ignore_errors='n' ) or is unspecified.</td>
</tr>
</tbody>
</table>
Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use ST_GeomFromWKB.

Convert GEOMETRY into WKT:

```sql
=> CREATE TABLE t(g GEOMETRY);
CREATE TABLE
=> INSERT INTO t VALUES(
  ST_GeomFromWKB(X'01030000000100000004000000000000000000000000000000000000000000f03f0000000000000000f64ae1c7022db544000000000000f03f00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
ST_Intersection

Calculates the set of points shared by two GEOMETRY objects.

Behavior Type

Immutable

Syntax

ST_Intersection( g1, g2 )

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g1</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
<tr>
<td>g2</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Intersection.

Two polygons intersect at a single point:

```sql
=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('POLYGON((0 2,1,0 -1, 0 2))'),ST_GeomFromText('POLYGON((-1 2,0 0,-2 0,-1 2)'))));
ST_AsText
-----------------
POINT(0 0)
(1 row)
```

Two polygons:
Two non-intersecting linestrings:

=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('POLYGON((1 2,1 5,4 5, 4 2,1 2))'), ST_GeomFromText('POLYGON((3 1,3 5,3 5,1,3 1))')));

ST_AsText
----------------------
POLYGON ((4 3, 4 2, 3 2, 3 3, 4 3))
(1 row)

=> SELECT ST_AsText(ST_Intersection(ST_GeomFromText('LINESTRING(1 1,1 3,3 3)'),
ST_GeomFromText('LINESTRING(1 5,1 7,-1 7)')));

ST_AsText
----------------------
GEOMETRYCOLLECTION EMPTY
(1 row)

ST_Intersects

Determines if two GEOMETRY or GEOGRAPHY objects intersect or touch at a single point. If ST_Disjoint returns TRUE, ST_Intersects returns FALSE for the same GEOMETRY or GEOGRAPHY objects.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable
Syntax

\texttt{ST_Intersects( g1, g2 )}
\[\text{[USING PARAMETERS } \text{bbox} = \{ \text{true} | \text{false} \}, \text{spheroid} = \{ \text{true} | \text{false} \}\text{]\]}

Arguments

<table>
<thead>
<tr>
<th>$g1$</th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g2$</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox = {true</td>
<td>false}</td>
</tr>
<tr>
<td>spheroid = {true</td>
<td>false}</td>
</tr>
</tbody>
</table>

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data Type</td>
<td>GEOGRAPHY (WGS84)</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>Point-Point</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Linestring-Point</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Polygon-Point</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multipolygon-Point</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how to use `ST_Intersects`.

**Two polygons do not intersect or touch:**

```sql
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1,2,0 3,0 1,-1 2))'),
                         ST_GeomFromText('POLYGON((1 0,1 1,2,1 0))'));
ST_Intersects
----------
f
(1 row)
```

**Two polygons touch at a single point:**

```sql
```
```sql
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
                      ST_GeomFromText('POLYGON((1 1,0 1 1,1 0))'));
ST_Intersects
----------
t
(1 row)
```

Two polygons intersect:

```sql
=> SELECT ST_Intersects (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),
                      ST_GeomFromText('POLYGON((0 2, -1 3, -2 0, 0 2))'));
ST_Intersects
----------
t
(1 row)
```
See Also

ST_Disjoint

ST_IsEmpty

Determines if a spatial object represents the empty set. An empty object has no dimension.

Behavior Type

Immutable

Syntax

ST_IsEmpty( g )

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
</table>

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

The following example shows how to use ST_IsEmpty.

An empty polygon:

```sql
=> SELECT ST_IsEmpty(ST_GeomFromText('GeometryCollection EMPTY'));
ST_IsEmpty
----------
t
(1 row)
```

ST_IsSimple

Determines if a spatial object does not intersect itself or touch its own boundary at any point.

Behavior Type

Immutable

Syntax

```
ST_IsSimple( g )
```

Arguments

| g  | Spatial object, type GEOMETRY or GEOGRAPHY |

Returns

BOOLEAN
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_IsSimple.

Polygon does not intersect itself:

```
=> SELECT ST_IsSimple(ST_GeomFromText('POLYGON((-1 2, 0 3, 1 2, -2, -1 2)))
ST_IsSimple
--------------
t
(1 row)
```

Linestring intersects itself:
SELECT ST_IsSimple(ST_GeographyFromText('LINESTRING(10 10, 25 25, 26 34.5, 10 30, 10 20, 20 10)'));

ST_IsSimple
--------------
f
(1 row)

Linestring touches its interior at one or more locations:

=> SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0, 0 1, 1 0, 0 1, 0 0)'));
ST_IsSimple
--------------
f
(1 row)

**ST_IsValid**

Determines if a spatial object is well formed or valid. If the object is valid, ST_IsValid returns TRUE; otherwise, it returns FALSE. Use STV_IsValidReason to identify the location of the invalidity.
Spatial validity applies only to polygons and multipolygons. A polygon or multipolygon is valid if all of the following are true:

- The polygon is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.
- The boundary does not touch or cross itself.
- Any polygons in the interior do not touch the boundary of the exterior polygon except at a vertex.


If you are not sure if a polygon is valid, run ST_IsValid first. If you pass an invalid spatial object to a Vertica Place function, the function fails or returns incorrect results.

Behavior Type

Immutable

Syntax

\[ \text{ST_IsValid}( g ) \]

Arguments

| \( g \) | Geospatial object to test for validity, value of type GEOMETRY or GEOGRAPHY (WGS84). |

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Point</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how to use `ST_IsValid`.

**Valid polygon:**

```sql
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1,1 3,3 3,3 1,1 1))'));
ST_IsValid
-----------
t
(1 row)
```

**Invalid polygon:**

```sql

```
=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 3,3 2,1 1,3 0,1 0,1 3))'));
ST_IsValid
----------
f
(1 row)

Invalid polygon:

=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,2 2,0,2,0,0,0,0,0))'));
ST_IsValid
----------
f
(1 row)

Invalid multipolygon:
=> SELECT ST_IsValid(ST_GeomFromText('MULTIPOLYGON(((0 0, 0 1, 1 1, 0 0),
(0.5 0.5, 0.7 0.5, 0.7 0.7, 0.5 0.7, 0.5 0.5)))'));
ST_IsValid
----------
f
(1 row)

Valid polygon with hole:

=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1 1,3,6,-1,0.5,-1,1,1),
(1,3,1,2,0,1,1))'));
ST_IsValid
----------
t
(1 row)

Invalid polygon with hole:

=> SELECT ST_IsValid(ST_GeomFromText('POLYGON((1,1,3,6,-1,0.5,-1,1,1),
(1,1,4.5,1,2,0,1,1))'));
ST_IsValid
----------
f
(1 row)
**ST_Length**

Calculates the length of a spatial object. For GEOMETRY objects, the length is measured in Cartesian coordinate units. For GEOGRAPHY objects, the length is measured in meters. Calculates the length as follows:

- The length of a point or multipoint object is 0.
- The length of a linestring is the sum of the lengths of each line segment. The length of a line segment is the distance from the start point to the end point.
- The length of a polygon is the sum of the lengths of the exterior boundary and any interior boundaries.
- The length of a multilinestring, multipolygon, or geometrycollection is the sum of the lengths of all the objects it contains.

Note: ST_Length does not calculate the length of WKTs or WKBs. To calculate the lengths of those objects, use the Vertica LENGTH SQL function with ST_AsBinary or ST_AsText.

**Behavior Type**

Immutable

**Syntax**

`ST_Length( g )`

**Arguments**

| `g` | Spatial object for which you want to calculate the length, type GEOMETRY or GEOGRAPHY |

**Returns**

FLOAT
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use `ST_Length`.

Returns length in Cartesian coordinate units:

```sql
=> SELECT ST_Length(ST_GeomFromText('LINESTRING(-1,-1,2,4,5,6)'));
ST_Length
-------------------
10.6766190873295
(1 row)
```

Returns length in meters:

```sql
=> SELECT ST_Length(ST_GeographyFromText('LINESTRING(-56.12 38.26,-57.51 39.78, -56.37 45.24)'));
ST_Length
-------------------
821580.025733461
(1 row)
```

**ST_NumGeometries**

Returns the number of geometries contained within a spatial object. Single GEOMETRY or GEOGRAPHY objects return 1 and empty objects return NULL.
Behavior Type

Immutable

Syntax

ST_NumGeometries( g )

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object of type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
</table>

Returns

INTEGER

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Examples

The following example shows how to use ST_NumGeometries.

Return the number of geometries:

```
=> SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING((1 5, 2 4, 5 3, 6 6), (3 5, 3 7)))');
ST_NumGeometries
-------------------
  2
(1 row)
```

See Also

ST_GeometryN

ST_NumPoints

Calculates the number of vertices of a spatial object, empty objects return NULL.

The first and last vertex of polygons and multipolygons are counted separately.

Behavior Type

Immutable

Syntax

```
ST_NumPoints( g )
```

Arguments

| g  | Spatial object for which you want to count the vertices, type GEOMETRY or GEOGRAPHY |
Returns

INTEGER

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_NumPoints.

Returns the number of vertices in a linestring:

```
=> SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(1.33 1.56,2.31 3.4,2.78 5.82,
  3.76 3.9,4.11 3.27,5.85 4.34,6.9 4.231,7.61 5.77)')));
ST_NumPoints
-------------
   8
(1 row)
```

Use ST_Boundary and ST_NumPoints to return the number of vertices of a polygon:

```
=> SELECT ST_NumPoints(ST_Boundary(ST_GeomFromText('POLYGON((1 2,1 4,
  2 5,3 6,4 6,5 4,3 3,1 2)))'));
ST_NumPoints
-------------
   9
(1 row)
```
ST_Overlaps

Determines if a GEOMETRY object shares space with another GEOMETRY object, but is not completely contained within that object. They must overlap at their interiors. If two objects touch at a single point or intersect only along a boundary, they do not overlap. Both parameters must have the same dimension; otherwise, ST_Overlaps returns FALSE.

Behavior Type
Immutable

Syntax
ST_Overlaps ( g1, g2 )

Arguments
\n
<table>
<thead>
<tr>
<th>g1</th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>

Returns
BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use `ST_Overlaps`.

Polygon_1 overlaps but does not completely contain Polygon_2:

```sql
=> SELECT ST_Overlaps(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 0 0))'),
                        ST_GeomFromText('POLYGON((0.5 0.5, 0.7 0.5, 0.7 0.7, 0.5 0.7, 0.5 0.5))'));
ST_Overlaps
  t
(1 row)
```

Two objects with different dimensions:

```sql
=> SELECT ST_Overlaps(ST_GeomFromText('LINESTRING(2 2,4 4)'),
                        ST_GeomFromText('POINT(3 3)'));
ST_Overlaps
  f
(1 row)
```

ST_PointFromGeoHash

Returns the center point of the specified GeoHash.

Behavior Type

Immutable
Syntax

`ST_PointFromGeoHash(GeoHash)`

Arguments

| GeoHash | A valid GeoHash string of arbitrary length. |

Returns

GEOGRAPHY POINT

Examples

The following examples show how to use `ST_PointFromGeoHash`.

Returns the geography point of a high-level GeoHash and uses `ST_AsText` to convert that point into Well-Known Text:

```sql
=> SELECT ST_AsText(ST_PointFromGeoHash('dr'));
ST_AsText
---------------------
POINT (-73.125 42.1875)
(1 row)
```

Returns the geography point of a detailed GeoHash and uses `ST_AsText` to convert that point into Well-Known Text:

```sql
=> SELECT ST_AsText(ST_PointFromGeoHash('1234567890bcdefhjkmn'));
ST_AsText
---------------------
POINT (-122.196077187 -88.2297377551)
(1 row)
```

**ST_PointN**

Finds the $n^{th}$ point of a spatial object. If you pass a negative number, zero, or a number larger than the total number of points on the linestring, `ST_PointN` returns NULL.

The vertex order is based on the Well-Known Text (WKT) representation of the spatial object.
Behavior Type

Immutable

Syntax

ST_PointN( g, n )

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object to search, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Point in the spatial object to be returned. The index is one-based, type INTEGER</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY or GEOGRAPHY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_PointN.

Returns the fifth point:

```sql
=> SELECT ST_AsText(ST_PointN(ST_GeomFromText('POLYGON(( 2 6, 2 9, 6 9, 7 7, 4 6, 2 6))'), 5));

ST_AsText
----------------
POINT (4 6)
(1 row)
```

Returns the second point:

```sql
=> SELECT ST_AsText(ST_PointN(ST_GeographyFromText('LINESTRING(23.41 24.93,34.2 32.98,40.7 41.19)'), 2));

ST_AsText
----------------
POINT (34.2 32.98)
(1 row)
```

ST_Relate

Determines if a given GEOMETRY object is spatially related to another GEOMETRY object, based on the specified DE-9IM pattern matrix string.

The DE-9IM standard identifies how two objects are spatially related to each other.

Behavior Type

Immutable

Syntax

```
ST_Relate( g1, g2, matrix )
```

Arguments

<table>
<thead>
<tr>
<th>$g1$</th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g2$</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>
matrix | DE-9IM pattern matrix string, type CHAR(9). This string represents a 3 x 3 matrix of restrictions on the dimensions of the respective intersections of the interior, boundary, and exterior of the two geometries. Must contain exactly 9 of the following characters:

- T
- F
- 0
- 1
- 2
- *

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_Relate.
The DE-9IM pattern for "equals" is 'T**FFF2':

```sql
=> SELECT ST_Relate(ST_GeomFromText('LINESTRING(0 1,2 2)'),
                   ST_GeomFromText('LINESTRING(2 2,0 1)'), 'T**FFF2');
ST_Relate
---------
t (1 row)
```

The DE-9IM pattern for "overlaps" is 'T*T***T**':

```sql
=> SELECT ST_Relate(ST_GeomFromText('POLYGON((-1 -1,0 1,2 2,-1 -1))'),
                   ST_GeomFromText('POLYGON((0 1,1,1,1,0 1))'), 'T*T***T**');
ST_Relate
---------
t (1 row)
```

**ST_SRID**

Identifies the spatial reference system identifier (SRID) stored with a spatial object.

The SRID of a GEOMETRY object can only be determined when passing an SRID to either ST_GeomFromText or ST_GeomFromWKB. ST_SRID returns this stored value. SRID values of 0 to 2^{32}-1 are valid.

**Behavior Type**

Immutable

**Syntax**

```sql
ST_SRID( g )
```

**Arguments**

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object for which you want the SRID, type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
</table>

**Returns**

INTEGER
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_SRID.

The default SRID of a GEOMETRY object is 0:

```sql
=> SELECT ST_SRID(ST_GeomFromText(
   'POLYGON((-1 -1,2,0,1,-1))'));
ST_SRID
---------
0
(1 row)
```

The default SRID of a GEOGRAPHY object is 4326:

```sql
=> SELECT ST_SRID(ST_GeographyFromText(
   'POLYGON((22 35,24 35,26 32,22 35))'));
ST_SRID
---------
4326
(1 row)
```
**ST_SymDifference**

Calculates all the points in two GEOMETRY objects except for the points they have in common, but including the boundaries of both objects.

This result is called the symmetric difference and is represented mathematically as: \( \text{Closure}(g_1 - g_2) \cup \text{Closure}(g_2 - g_1) \)

**Behavior Type**

Immutable

**Syntax**

\[ \text{ST}_\text{SymDifference}(g_1, g_2) \]

**Arguments**

| \( g_1 \) | Spatial object, type GEOMETRY |
| \( g_2 \) | Spatial object, type GEOMETRY |

**Returns**

GEOMETRY

**Supported Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_SymDifference.

Returns the two linestrings:

```
=> SELECT ST_AsText(ST_SymDifference(ST_GeomFromText('LINESTRING(30 40, 30 55)'), ST_GeomFromText('LINESTRING(30 32.5, 30 47.5)')));
  ST_AsText
-------------------------
MULTILINESTRING ((30 47.5, 30 55), (30 32.5, 30 40))
(1 row)
```

Returns four squares:
ST_Touches

Determines if two GEOMETRY objects touch at a single point or along a boundary, but do not have interiors that intersect.

GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

Behavior Type

Immutable

Syntax

```
ST_Touches( g1, g2 )

[USING PARAMETERS spheroid={true | false}]
```

Arguments

<table>
<thead>
<tr>
<th>g1</th>
<th>Spatial object, value of type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2</td>
<td>Spatial object, value of type GEOMETRY</td>
</tr>
</tbody>
</table>
Parameters

`spheroid = {true | false}` (Optional) BOOLEAN that specifies whether to use a perfect sphere or WGS84.

Default: False

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatible GEOGRAPHY pairs:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-Point</td>
<td>No</td>
</tr>
<tr>
<td>Linestring-Point</td>
<td>No</td>
</tr>
<tr>
<td>Polygon-Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon-Point</td>
<td>No</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_Touches.

Two polygons touch at a single point:

```sql
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
   ST_GeomFromText('POLYGON((1 3,0 3,1 2,1 3)')));
ST_Touches
----------
t
(1 row)
```

Two polygons touch only along part of the boundary:

```sql
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
   ST_GeomFromText('POLYGON((1 2,0 3,0 1,1 2)')));
ST_Touches
----------
t
(1 row)
```

Two polygons do not touch at any point:

```sql
=> SELECT ST_Touches(ST_GeomFromText('POLYGON((-1 2,0 3,0 1,-1 2))'),
   ST_GeomFromText('POLYGON((0 2,-1 3,-2 0,0 2)')));
ST_Touches
----------
f
(1 row)
```

ST_Transform

Returns a new GEOMETRY with its coordinates converted to the spatial reference system identifier (SRID) used by the srId argument.

This function supports the following transformations:

- EPSG 4326 (WGS84) to EPSG 3857 (Web Mercator)
- EPSG 3857 (Web Mercator) to EPSG 4326 (WGS84)

For EPSG 4326 (WGS84), unless the coordinates fall within the following ranges, conversion results in failure:
• Longitude limits: -572 to +572
• Latitude limits: -89.9999999 to +89.9999999

Behavior Type

Immutable

Syntax

\texttt{ST\_Transform( g1, srid )}

Arguments

\begin{tabular}{|l|l|}
\hline
\textit{g1} & Spatial object of type GEOMETRY. \\
\hline
\textit{srid} & Spatial reference system identifier (SRID) to which you want to convert your spatial object, of type INTEGER. \\
\hline
\end{tabular}

Returns

GEOMETRY

Supported Data Types

\begin{tabular}{|l|c|c|c|}
\hline
Data Type & GEOMETRY & GEOGRAPHY (Perfect Sphere) & GEOGRAPHY (WGS84) \\
\hline
Point & Yes & No & No \\
Multipoint & Yes & No & No \\
Linestring & Yes & No & No \\
Multilinestring & Yes & No & No \\
Polygon & Yes & No & No \\
\hline
\end{tabular}
Examples

The following example shows how you can transform data from Web Mercator (3857) to WGS84 (4326):

```sql
=> SELECT ST_AsText(ST_Transform(STV_GeometryPoint(7910240.56433, 5215074.23966, 3857), 4326));
---
POINT (71.0589 42.3601)
(1 row)
```

The following example shows how you can transform linestring data in a table from WGS84 (4326) to Web Mercator (3857):

```sql
=> CREATE TABLE transform_line_example (g GEOMETRY);
CREATE TABLE
=> COPY transform_line_example (gx FILLER LONG VARCHAR, g AS ST_GeomFromText(gx, 4326)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
> LINESTRING(0 0, 1 1, 2 2, 3 4)
> \.
=> SELECT ST_AsText(ST_Transform(g, 3857)) FROM transform_line_example;
---
LINESTRING (0 -7.08115455161e-10, 111319.490793 111325.142866, 222638.981587 222684.208506, 333958.47238 445640.109656)
(1 row)
```

The following example shows how you can transform point data in a table from WGS84 (4326) to Web Mercator (3857):

```sql
=> CREATE TABLE transform_example (x FLOAT, y FLOAT, srid INT);
CREATE TABLE
=> COPY transform_example FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
> 42.3601|71.0589|4326
> 122.4194|37.7749|4326
> 94.5786|39.0997|4326
> \.
=> SELECT ST_AsText(ST_Transform(STV_GeometryPoint(x, y, srid), 3857)) FROM transform_example;
---
POINT (4715504.76195 11422441.5961)
POINT (13627665.2712 4547675.35434)
POINT (18528441.5919 4735962.8206)
(3 rows)
```
ST_Union

Calculates the union of all points in two spatial objects. This result is represented mathematically by: \( g1 \cup g2 \)

Behavior Type

Immutable

Syntax

\[ \text{ST}_\text{Union}( \textit{g1}, \textit{g2} ) \]

Arguments

<table>
<thead>
<tr>
<th>( g1 )</th>
<th>Spatial object, type GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g2 )</td>
<td>Spatial object, type GEOMETRY</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

The following example shows how to use ST_Union.

Returns a polygon that represents all the points contained in these two polygons:

```sql
=> SELECT ST_AsText(ST_Union(ST_GeomFromText('POLYGON((0 2,1 1,-1,-1 1,0 2))'),
                               ST_GeomFromText('POLYGON((-1 2, 0 0, -2 0, -1 2))')));
               ST_AsText
--------------------------
POLYGON ((0 2, 1 1, 0 -1, 0.5 0, -2 0, -1 2, -0.666666666667 1.3333333333, 0 2))
(1 row)
```

**ST_Within**

If spatial object \(g1\) is completely inside of spatial object \(g2\), then \(ST_{\text{Within}}\) returns true. Both parameters must be the same spatial data type. Either specify two GEOMETRY objects or two GEOGRAPHY objects.

If an object such as a point or linestring only exists along a polygon's boundary, then \(ST_{\text{Within}}\) returns false. The interior of a linestring is all the points along the linestring except the start and end points.

\(ST_{\text{Within}}(g1, g2)\) is functionally equivalent to \(ST_{\text{Contains}}(g2, g1)\).
GEOGRAPHY Polygons with a vertex or border on the International Date Line (IDL) or the North or South pole are not supported.

**Behavior Type**

Immutable

**Syntax**

```
ST_Within( g1, g2
               [USING PARAMETERS spheroid={true | false}] )
```

**Arguments**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$g1$</td>
<td>Spatial object, type GEOMETRY or GEOGRAPHY</td>
</tr>
<tr>
<td>$g2$</td>
<td>Spatial object, type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>spheroid = (true</td>
<td>false)</td>
</tr>
</tbody>
</table>

**Returns**

BOOLEAN

**Supported Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Geometry Type</td>
<td>Compatible with Perfect Sphere</td>
<td>Compatible with WGS84</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatible GEOGRAPHY pairs:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-Point</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Point-Linestring</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Point-Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Point-Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_Within.

The first polygon is completely contained within the second polygon:

```sql
=> SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 0 -1,0 2))'),
  ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
ST_Within
---------
t
(1 row)
```

The point is on a vertex of the polygon, but not in its interior:

```sql
=> SELECT ST_Within(ST_GeographyFromText('POINT(30 25)'),
  ST_GeographyFromText('POLYGON((25 25,25 35,32.2 35,30 25,25 25))'));
ST_Within
---------
f
(1 row)
```

Two polygons are spatially equivalent:
=> SELECT ST_Within (ST_GeomFromText('POLYGON((-1 2, 0 3, 0 1, -1 2))'),
    ST_GeomFromText('POLYGON((0 3, -1 2, 0 1, 0 3))'));

ST_Within
----------
t
(1 row)

See Also

- **ST_Contains**
- **ST_Overlaps**

**ST_X**

Determines the x-coordinate for a GEOMETRY point or the longitude value for a GEOGRAPHY point.

**Behavior Type**

Immutable

**Syntax**

`ST_X( g )`

**Arguments**

| g   | Point of type GEOMETRY or GEOGRAPHY |

**Returns**

FLOAT
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_X.

Returns the x-coordinate:

```
=> SELECT ST_X(ST_GeomFromText('POINT(3.4 1.25)'));
  ST_X
-----
  3.4
(1 row)
```

Returns the longitude value:

```
=> SELECT ST_X(ST_GeographyFromText('POINT(25.34 45.67)'));
  ST_X
-----
  25.34
(1 row)
```

ST_XMax

Returns the maximum x-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.
For GEOGRAPHY types, Vertica Place computes maximum coordinates by calculating the maximum longitude of the great circle arc from \((\text{MAX(longitude)}, \text{ST_YMin(GEOGRAPHY)})\) to \((\text{MAX(longitude)}, \text{ST_YMax(GEOGRAPHY)})\). In this case, \(\text{MAX(longitude)}\) is the maximum longitude value of the geography object.

If either latitude or longitude is out of range, \(\text{ST_XMax}\) returns the maximum plain value of the geography object.

**Behavior Type**

Immutable

**Syntax**

\[
\text{ST_XMax( } g \text{ )}
\]

**Arguments**

| \(g\) | Spatial object for which you want to find the maximum x-coordinate, type GEOMETRY or GEOGRAPHY. |

**Returns**

FLOAT

**Supported Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_XMax.

Returns the maximum x-coordinate within a rectangle:

```sql
=> SELECT ST_XMax(ST_GeomFromText('POLYGON((0 1,0 2,1 2,1 1,0 1))'));
ST_XMax
--------
 1
(1 row)
```

Returns the maximum longitude value within a rectangle:

```sql
=> SELECT ST_XMax(ST_GeographyFromText(
    'POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))'));
ST_XMax
--------
-71
(1 row)
```

**ST_XMin**

Returns the minimum x-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes minimum coordinates by calculating the minimum longitude of the great circle arc from (MIN(longitude), ST_YMin(GEOGRAPHY)) to (MIN(longitude), ST_YMax(GEOGRAPHY)). In this case, MIN(latitude) represents the minimum longitude value of the geography object.

If either latitude or longitude is out of range, ST_XMin returns the minimum plain value of the geography object.

**Behavior Type**

Immutable
Syntax

\texttt{ST\_XMin( g )}

Arguments

| g     | Spatial object for which you want to find the minimum x-coordinate, type GEOMETRY or GEOGRAPHY. |

Returns

FLOAT

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use \texttt{ST\_XMin}.

Returns the minimum x-coordinate within a rectangle:
=> SELECT ST_XMin(ST_GeomFromText('POLYGON((0 1,0 2,1 2,1 1,0 1))'));
ST_XMin
---------
 0
(1 row)

Returns the minimum longitude value within a rectangle:

=> SELECT ST_XMin(ST_GeographyFromText('POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35))'));
ST_XMin
---------
 -71.5
(1 row)

**ST_YMax**

Returns the maximum y-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Places computes maximum coordinates by calculating the maximum latitude of the great circle arc from (ST_XMin(GEOGRAPHY), MAX(latitude)) to (ST_XMax(GEOGRAPHY), MAX(latitude)). In this case, MAX(latitude) is the maximum latitude value of the geography object.

If either latitude or longitude is out of range, ST_YMax returns the maximum plain value of the geography object.

**Behavior Type**

Immutable

**Syntax**

`ST_YMax( g )`

**Arguments**

| g  | Spatial object for which you want to find the maximum y-coordinate, type GEOMETRY or GEOGRAPHY. |
Returns

FLOAT

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how to use ST_YMax.

Returns the maximum y-coordinate within a rectangle:

```sql
=> SELECT ST_YMax(ST_GeomFromText('POLYGON((0 1,0 4,1 4,1 1,0 1)'));

ST_YMax
---------
4
(1 row)
```

Returns the maximum latitude value within a rectangle:

```sql
=> SELECT ST_YMax(ST_GeographyFromText('POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35)')));

ST_YMax
---------
42.3802715689979
(1 row)
```
**ST_YMin**

Returns the minimum y-coordinate of the minimum bounding rectangle of the GEOMETRY or GEOGRAPHY object.

For GEOGRAPHY types, Vertica Place computes minimum coordinates by calculating the minimum latitude of the great circle arc from (ST_XMin(GEOGRAPHY), MIN(latitude)) to (ST_XMax(GEOGRAPHY), MIN(latitude)). In this case, MIN(latitude) represents the minimum latitude value of the geography object.

If either latitude or longitude is out of range, ST_YMin returns the minimum plain value of the geography object.

**Behavior Type**

Immutable

**Syntax**

```
ST_YMin( g )
```

**Arguments**

| g | Spatial object for which you want to find the minimum y-coordinate, type GEOMETRY or GEOGRAPHY. |

**Returns**

FLOAT

**Supported Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Examples

The following examples show how to use ST_YMin.

Returns the minimum y-coordinate within a rectangle:

```sql
=> SELECT ST_YMin(ST_GeomFromText('POLYGON((0 1,0 4,1 4,1 1,0 1))'));
ST_YMin
-----------
 1
(1 row)
```

Returns the minimum latitude value within a rectangle:

```sql
=> SELECT ST_YMin(ST_GeographyFromText('POLYGON((-71.50 42.35, -71.00 42.35, -71.00 42.38, -71.50 42.38, -71.50 42.35'))));
ST_YMin
-----------
 42.35
(1 row)
```

**ST_Y**

Determines the y-coordinate for a GEOMETRY point or the latitude value for a GEOGRAPHY point.

**Behavior Type**

Immutable
Syntax

\texttt{ST_Y( g )}

Arguments

\begin{tabular}{|l|l|}
\hline
\texttt{g} & Point of type GEOMETRY or GEOGRAPHY \\
\hline
\end{tabular}

Returns

FLOAT

Supported Data Types

\begin{tabular}{|l|c|c|c|}
\hline
Data Type & GEOMETRY & GEOGRAPHY (Perfect Sphere) & GEOGRAPHY (WGS84) \\
\hline
Point & Yes & Yes & Yes \\
Multipoint & No & No & No \\
Linestring & No & No & No \\
Multipolilinestring & No & No & No \\
Polygon & No & No & No \\
Multipolygon & No & No & No \\
GeometryCollection & No & No & No \\
\hline
\end{tabular}

Examples

The following examples show how to use \texttt{ST_Y}.

Returns the \texttt{y-coordinate}:
=> SELECT ST_Y(ST_GeomFromText('POINT(3.25)'));
ST_Y
------
5.25
(1 row)

Returns the latitude value:

=> SELECT ST_Y(ST_GeographyFromText('POINT(35.44 51.04)'));
ST_Y
------
51.04
(1 row)

**STV_AsGeoJSON**

Returns the geometry or geography argument as a Geometry Javascript Object Notation (GeoJSON) object.

**Behavior Type**

Immutable

**Syntax**

```sql
STV_AsGeoJSON( g, [USING PARAMETERS maxdecimals=[dec_value]])
```

**Arguments**

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object of type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxdecimals = dec_value</td>
<td>(Optional) Integer value. Determines the maximum number of digits to output after the decimal of floating point coordinates. Valid values: Between 0 and 15. Default value: 6</td>
</tr>
</tbody>
</table>

**Returns**

LONG VARCHAR
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how you can use STV_AsGeoJSON.

Convert a geometry polygon to GeoJSON:

```sql
=> SELECT STV_AsGeoJSON(ST_GeomFromText('POLYGON((3 2, 4 3, 5 1, 3 2), (3.5 2, 4 2.5, 4.5 1.5, 3.5 2))'));

STV_AsGeoJSON
-------------------
{"type":"Polygon","coordinates":
[[[3,2],[4,3],[5,1],[3,2]],
[[3.5,2],[4,2.5],[4.5,1.5],[3.5,2]]]}

(1 row)
```

Convert a geography point to GeoJSON:

```sql
=> SELECT STV_AsGeoJSON(ST_GeographyFromText('POINT(42.36011 71.05899)') USING PARAMETERS maxdecimals=4);

STV_AsGeoJSON
---------------------
{"type":"Point","coordinates":[42.3601,71.059]}

(1 row)
```

STV_Create_Index

Creates a spatial index on a set of polygons to speed up spatial intersection with a set of points.
A spatial index is created from an input polygon set, which can be the result of a query. Spatial indexes are created in a global name space. Vertica uses a distributed plan whenever the input table or projection is segmented across nodes of the cluster.

The OVER() clause must be empty.

Important: You cannot access spatial indexes on newly added nodes without rebalancing your cluster. For more information, see REBALANCE_Cluster.

Behavior Type

Immutable

Note: Indexes are not connected to any specific table. Subsequent DML commands on the underlying table or tables of the input data source do not modify the index.

Syntax

STV_Create_Index( gid, g
  USING PARAMETERS index='index_name'
  [, overwrite={ true | false }] 
  [, max_mem_mb=maxmem_value]
  [, skip_nonindexable_polygons={true | false} ]
  OVER()
  [ AS (polygons, srid, min_x, min_y, max_x, max_y, info) ]

Arguments

<table>
<thead>
<tr>
<th>gid</th>
<th>Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Name of a geometry or geography (WGS84) column or expression that contains polygons and multipolygons. Only polygon and multipolygon can be indexed. Other shape types are excluded from the index.</td>
</tr>
</tbody>
</table>

Parameters

| index = 'index_name' | Name of the index, type VARCHAR. Index names cannot exceed 110 |
characters. The slash, backslash, and tab characters are not allowed in index names.

| overwrite = [ true | false ] | (Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL. Default: False |
| --- | --- |

| max_mem_mb = maxmem_value | (Optional) A positive integer that assigns a limit to the amount of memory in megabytes that STV_Create_Index can allocate during index construction. On a multi-node database this is the memory limit per node. The default value is 256. Do not assign a value higher than the amount of memory in the GENERAL resource pool. For more information about this pool, see Using Queries to Monitor Resource Pool Size and Usage. Setting a value for max_mem_mb that is at or near the maximum memory available on the node can negatively affect your system's performance. For example, it could cause |
other queries to time out waiting for memory resources during index construction.

\[
\text{skip\_nonindexable\_polygons} = [\text{true} | \text{false}]
\]

(Optional) BOOLEAN
In rare cases, intricate polygons (for instance, with too high resolution or anomalous spikes) cannot be indexed. These polygons are considered non-indexable. When set to False, non-indexable polygons cause the index creation to fail. When set to True, index creation can succeed by excluding non-indexable polygons from the index.

To review the polygons that were not able to be indexed, use STV_Describe_Index with the parameter list_polygon. Default: False

## Returns

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{polygons}</td>
<td>Number of polygons indexed.</td>
</tr>
<tr>
<td>\textit{SRID}</td>
<td>Spatial reference system identifier.</td>
</tr>
<tr>
<td>\textit{min_x, min_y, max_x, max_y}</td>
<td>Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. ((\text{min_x, min_y})) are the southwest coordinates, and ((\text{max_x, max_y})) are the north-east coordinates.</td>
</tr>
</tbody>
</table>
info

Lists the number of excluded spatial objects as well as their type that were excluded from the index.

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

Recommendations

- Segment large polygon tables across multiple nodes. Table segmentation causes index creation to run in parallel, leveraging the Massively Parallel Processing (MPP) architecture in Vertica. This significantly reduces execution time on large tables.

  Vertica recommends that you segment the table from which you are building the index when the total number of polygons is large.

- STV_Create_Index can consume large amounts of processing time and memory.
Vertica recommends that when indexing new data for the first time, you monitor memory usage to be sure it stays within safe limits. Memory usage depends on number of polygons, number of vertices, and the amount of overlap among polygons.

- STV_Create_Index tries to allocate memory before it starts creating the index. If it cannot allocate enough memory, the function fails. If not enough memory is available, try the following:
  - Create the index at a time of less load on the system.
  - Avoid concurrent index creation.
  - Try segmenting the input table across the nodes of the cluster.

- Ensure that all of the polygons you plan to index are valid polygons. STV_Create_Index and STV_Refresh_Index do not check polygon validity when building an index.

For more information, see Ensuring Polygon Validity Before Creating or Refreshing an Index.

**Limitations**

- Any indexes created prior to 9.0.x need to re-created.

- Index creation fails if there are WGS84 polygons with vertices on the International Date Line (IDL) or the North and South Poles.

- The backslash or tab characters are not allowed in index names.

- Indexes cannot have names greater than 110 characters.

- The following geometries are excluded from the index:
  - Non-polygons
  - Geometries with NULL identifiers
  - NULL (multi) polygon
  - EMPTY (multi) polygon
  - Invalid (multi) polygon
The following geographies are excluded from the index:
- Polygons with holes
- Polygons crossing the International Date Line
- Polygons covering the north or south pole
- Antipodal polygons

Usage Tips
- To cancel an STV_Create_Index run, use Ctrl + C.
- If there are no valid polygons in the geom column, STV_Create_Index reports an error in vertica.log and stops index creation.
- If index creation uses a large amount of memory, consider segmenting your data to utilize parallel index creation.

Examples
The following examples show how to use STV_Create_Index.

Create an index with a single literal argument:

```sql
=> SELECT STV_Create_Index(1, ST_GeomFromText('POLYGON((0 0,15.2,3.9 15.2,3.9 0,0 0))
    USING PARAMETERS index='my_polygon') OVER();
d+| polygons | SRID | min_x | min_y | max_x | max_y | info
-------------|-------|-------|-------|-------|-------|-------
 1 | 0 | 0 | 0 | 3.9 | 15.2 | 
(1 row)
```

Create an index from a table:

```sql
=> CREATE TABLE polys (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY polys(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1|POLYGON((-31 74,8 70,8 50,-36 53,-31 74))
>> 2|POLYGON((-38 50,4 13,11 45,0 65,-38 50))
>> 3|POLYGON((10 20,15 60,20 45,46 15,10 20))
>> 4|POLYGON((5 20,9 30,20 45,36 35,5 20))
>> 5|POLYGON((12 23,9 38,20 45,36 35,37 67,45 80,50 20,12 23))
>> 
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true,
    max_mem_mb=256) OVER() FROM polys;
```
Create an index in parallel from a partitioned table:

```sql
=> CREATE TABLE pols (p INT, gid INT, geom GEOMETRY(1000)) SEGMENTED BY HASH(p) ALL NODES;
=> COPY pols (p, gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
>> 1|10|POLYGON((-31 74, 70, 8 50, -36 53, -31 74))
>> 1|11|POLYGON((-38 50, 4 13, 11 45, 0 65, -38 50))
>> 3|12|POLYGON((-12 42, -12 42, 27 48, 14 26, -12 42))
>> .
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons', overwrite=true, max_mem_mb=256) OVER() FROM pols;
```

## See Also

- **Spatial Joins with ST_Intersects and STV_Intersect**
- **STV_Intersect Scalar Function**
- **STV_Intersect Transform Function**
- **STV_Describe_Index**
- **STV_Drop_Index**
- **STV_Rename_Index**
- **Ensuring Polygon Validity Before Creating or Refreshing an Index**

### STV_Describe_Index

Retrieves information about an index that contains a set of polygons. If you do not pass any parameters, STV_Describe_Index returns all of the defined indexes.

The OVER() clause must be empty.
Behavior Type

Immutable

Syntax

STV_Describe_Index ( [ USING PARAMETERS [index='index_name']
                        [, list_polygons={true | false }] ] ) OVER ()

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index = 'index_name'</td>
<td>Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names.</td>
</tr>
<tr>
<td>list_polygon</td>
<td>(Optional) BOOLEAN that specifies whether to list the polygons in the index. The index argument must be used with this argument.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygons</td>
<td>Number of polygons indexed.</td>
</tr>
<tr>
<td>SRID</td>
<td>Spatial reference system identifier.</td>
</tr>
<tr>
<td>min_x, min_y, max_x, max_y</td>
<td>Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. (min_x, min_y) are the south-west coordinates, and (max_x, max_y) are the north-east coordinates.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the spatial index(es).</td>
</tr>
<tr>
<td>gid</td>
<td>Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.</td>
</tr>
<tr>
<td>state</td>
<td>The spatial object's state in the index. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• INDEXED - The spatial object was successfully indexed.</td>
</tr>
<tr>
<td></td>
<td>• SELF_INTERSECT - (WGS84 Only) The spatial object was not</td>
</tr>
</tbody>
</table>
indexed because one of its edges intersects with another of its edges.

- **EDGE CROSS IDL** - (WGS84 Only) The spatial object was not indexed because one of its edges crosses the International Date Line.

- **EDGE_HALF_CIRCLE** - (WGS84 Only) The spatial object was not indexed because it contains two adjacent vertices that are antipodal.

- **NON_INDEXABLE** - The spatial object was not able to be indexed.

<table>
<thead>
<tr>
<th>geography</th>
<th>The Well-Known Binary (WKB) representation of the spatial object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometry</td>
<td>The Well-Known Binary (WKB) representation of the spatial object.</td>
</tr>
</tbody>
</table>

**Privileges**

Any user with access to the STV_ * _Index functions can describe, rename, or drop indexes created by any other user.

**Limitations**

Some functionality will require the index to be rebuilt if the index was created with 9.0.x or earlier.

**Examples**

The following examples show how to use STV_Describe_Index.

Retrieve information about the index:

```sql
=> SELECT STV_Describe_Index (USING PARAMETERS index='my_polygons') OVER ();

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

(1 row)
```
Vertica Documentation
SQL Reference Manual
Return the names of all the defined indexes:
=> SELECT STV_Describe_Index() OVER ();
name
-----------------MA_counties_index
my_polygons
NY_counties_index
US_States_Index
(4 rows)

Return the polygons included in an index:
=> SELECT STV_Describe_Index(USING PARAMETERS index='my_polygons', list_polygons=TRUE) OVER ();
gid |
state
|
geometry
-----+---------------+---------------------------------12 | INDEXED
| \260\000\000\000\000\000\000\ ...
14 | INDEXED
| \200\000\000\000\000\000\000\ ...
10 | NON_INDEXABLE | \274\000\000\000\000\000\000\ ...
11 | INDEXED
| \260\000\000\000\000\000\000\ ...
(4 rows)

See Also
l

Spatial Joins with ST_Intersects and STV_Intersect

l

STV_Intersect Scalar Function

l

STV_Intersect Transform Function

l

STV_Drop_Index

l

STV_Rename_Index

STV_Drop_Index
Deletes a spatial index. If STV_Drop_Index cannot find the specified spatial index, it returns an
error.
The OVER clause must be empty.

Behavior Type
Immutable

Vertica Analytic Database (9.0.x)

Page 2588 of 6180


Syntax

STV_Drop_Index( USING PARAMETERS index = 'index_name' ) OVER ()

Arguments

| index = 'index_name' | Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names. |

Example

The following example shows how to use STV_Drop_Index.

Drop an index:

```sql
=> SELECT STV_Drop_Index(USING PARAMETERS index = 'my_polygons') OVER ();
   drop_index
         ----
   Index dropped
       (1 row)
```

See Also

- Spatial Joins with ST_Intersects and STV_Intersect
- STV_Create_Index
- STV_Describe_Index
- STV_Rename_Index
- STV_Intersect Scalar Function
- STV_Intersect Transform Function
STV_DWithin

Determines if the shortest distance from the boundary of one spatial object to the boundary of another object is within a specified distance.

Parameters $g1$ and $g2$ must be both GEOMETRY objects or both GEOGRAPHY objects.

Behavior Type

Immutable

Syntax

STV_DWithin( $g1$, $g2$, $d$ )

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g1$</td>
<td>Spatial object of type GEOMETRY or GEOGRAPHY</td>
</tr>
<tr>
<td>$g2$</td>
<td>Spatial object of type GEOMETRY or GEOGRAPHY</td>
</tr>
<tr>
<td>$d$</td>
<td>Value of type FLOAT indicating a distance. For GEOMETRY objects, the distance is measured in Cartesian coordinate units. For GEOGRAPHY objects, the distance is measured in meters.</td>
</tr>
</tbody>
</table>

Returns

BOOLEAN

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Type</td>
<td>GEOGRAPHY (Perfect Sphere)</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>Point-Point</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Point-Linestring</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Point-Polygon</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Point-Multilinestring</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Point-Multipolygon</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how to use STV_DWithin.

Two geometries are one Cartesian coordinate unit from each other at their closest points:

```sql
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2,0,1,-1))'),
                       ST_GeomFromText('POLYGON((4,3,2,3,4,3))'),1);
STV_DWithin
-----------
t
(1 row)
```

If you reduce the distance to 0.99 units:

```sql
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2,0,1,-1))'),
                       ST_GeomFromText('POLYGON((4,3,2,3,4,3))'),0.99);  
STV_DWithin
-----------
f
(1 row)
```

The first polygon touches the second polygon:
The first polygon is not within 1000 meters from the second polygon:

```sql
=> SELECT STV_DWithin(ST_GeomFromText('POLYGON((-1 -1,2,0 1,-1 -1))'),
                       ST_GeomFromText('POLYGON((1 1,2 3,4 5,1 1))'),0.00001);
```

```
t
(1 row)
```

**STV_Export2Shapefile**

Exports GEOGRAPHY or GEOMETRY data from a database table or a subquery to a shapefile. Writes the output to the directory specified using `STV_SetExportShapefileDirectory`.

**Behavior Type**

Immutable

**Syntax**

```sql
STV_Export2Shapefile( columns
                     USING PARAMETERS shapefile = 'name_of_shapefile'
                     [, overwrite = { TRUE | FALSE } ]
                     [, shape = ' { Point | Polygon | Linestring | Multipoint |
                           Multipolygon | Multilinestring } ' ]
                     OVER() )
```

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shapefile = 'name_of_shapefile'</td>
<td>Prefix of the component names of the shapefile, type VARCHAR. Must be unique.</td>
</tr>
</tbody>
</table>
end with the file extension .shp. Limited to 128 octets in length. For example, city-data.shp.

If you want to save the shapefile to a sub-directory you can do so by concatenating the sub-directory to name_of_shapefile. For example, visualizations/city-data.shp.

| overwrite = { TRUE | FALSE } |
|-----------------------------|
| (Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL. Default: False
| Overwriting may corrupt the existing files. |

| shape = '{ Point | Polygon | Linestring | Multipoint | Multipolygon | Multilinestring }' |
|-----------------------|
| Must be one of the following spatial classes: Point, Polygon, Linestring, Multipoint, Multipolygon, |
Multilinestring. Polygons and multipolygons always have a clockwise orientation. Default: Polygon

Arguments

| columns | The columns to export to the shapefile. A value of asterisk (*) is the equivalent to listing all columns of the FROM clause. |

Returns

Three files in the shapefile export directory with the extensions .shp, .shx, and .dbf.

Limitations

- If a multipolygon, multilinestring, or multipoint contains only one element, then it is written as a polygon, line, or point, respectively.
- Column names longer than 10 characters are truncated.
- Empty POINTS cannot be exported.
- All rows with NULL geometry or geography data are skipped.
- Unsupported or invalid dates are replaced with NULLs.
- Numeric values may lose precision when they are exported. This loss occurs because the target field in the .dbf file is a 64-bit FLOAT column, which can only represent about 15 significant digits.
Examples

The following example shows how you can use STV_Export2Shapefile to export all columns from the table geo_data to a shapefile named city-data.shp:

```sql
=> SELECT STV_Export2Shapefile(*
    USING PARAMETERS shapefile = 'visualizations/city-data.shp',
    overwrite = true, shape = 'Point')
OVER()
FROM geo_data
WHERE REVENUE > 25000;
```

<table>
<thead>
<tr>
<th>Rows Exported</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>6442892</td>
<td>v_geo-db_node0001: /home/geo/temp/visualizations/city-data.shp</td>
</tr>
</tbody>
</table>

(1 row)

STV_Extent

Returns a bounding box containing all of the input data.

Use STV_Extent inside of a nested query for best results. The OVER clause must be empty.

Important: STV_Extent does not return a valid polygon when the input is a single point.

Behavior Type

Immutable

Syntax

```
STV_Extent( g )
```

Arguments

| g   | Spatial object, type GEOMETRY. |

Returns

GEOMETRY
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples

The following examples show how you can use STV_Extent.

Return the bounding box of a linestring, and verify that it is a valid polygon:

```
=> SELECT ST_AsText(geom) AS bounding_box, ST_IsValid(geom)
    FROM (SELECT STV_Extent(ST_GeomFromText('LineString(0 0, 1 1)')) OVER() AS geom) AS g;
```

```
POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)) | t
(1 row)
```

Return the bounding box of spatial objects in a table:

```
=> CREATE TABLE misc_geo_shapes (id IDENTITY, geom GEOMETRY);
CREATE TABLE
=> COPY misc_geo_shapes (gx FILLER LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> POINT(-71.03 42.37)
=> LINESTRING(-71.058849 42.367501, -71.062240 42.371276, -71.067938 42.371246)
=> POLYGON((-71.066030 42.380617, -71.055827 42.376011, -71.060811 42.376734, -71.066030 42.380617))
=> .
=> SELECT ST_AsText(geom_col) AS bounding_box
    FROM (SELECT STV_Extent(geom) OVER() AS geom_col FROM misc_geo_shapes) AS g;
```

```
```

---

Vertica Analytic Database (9.0.x)
STV_ForceLHR

Alters the order of the vertices of a spatial object to follow the left-hand-rule.

Behavior Type
Immutable

Syntax

STV_ForceLHR( g, [USING PARAMETERS skip_nonreorientable_polygons={true | false} ])

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object, type GEOGRAPHY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip_nonreorientable_polygons = { true</td>
<td>false }</td>
</tr>
</tbody>
</table>

When set to False, non-orientable polygons generate an error. For example, if you use STV_ForceLHR or STV_Reverse with skip_nonreorientable_polygons set to False, a geography polygon containing a hole generates an error. When set to True, the result returned is the polygon, as passed to the API, without alteration.

This argument can help you when you are creating an index from a table containing polygons that cannot be re-oriented.

Vertica Place considers these polygons non-orientable:

- Polygons with a hole
Multipolygons

- Multipolygons
- Multipolygons with a hole

Default value: False

Returns

GEOGRAPHY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following example shows how you can use STV_ForceLHR.

Re-orient a geography polygon to left-hand orientation:

```sql
=> SELECT ST_AsText(STV_ForceLHR(ST_GeographyFromText('Polygon((1 1, 3 1, 2 2, 1 1))')));

POLYGON ((1 1, 3 1, 2 2, 1 1))
(1 row)
```

Reverse the orientation of a geography polygon by forcing left-hand orientation:
See Also

`STV_Revert`

**STV_Geography**

Casts a GEOMETRY object into a GEOGRAPHY object. The SRID value does not affect the results of Vertica Place queries.

When `STV_Geography` converts a GEOMETRY object to a GEOGRAPHY object, it sets its SRID to 4326.

**Behavior Type**

Immutable

**Syntax**

`STV_Geography( geom )`

**Arguments**

| `geom` | Spatial object that you want to cast into a GEOGRAPHY object, type GEOMETRY |

**Returns**

GEOGRAPHY
Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use STV_Geography.

To calculate the centroid of the GEOGRAPHY object, convert it to a GEOMETRY object, then convert it back to a GEOGRAPHY object:

```sql
=> CREATE TABLE geogs(g GEOGRAPHY);
CREATE TABLE
=> COPY geogs(gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> MULTIPΟINT(-108.619726 45.000284,-107.866813 45.00107,-106.363711 44.994223,-70.847746 41.205814)
>> .
=> SELECT ST_AsText(STV_Geography(ST_Centroid(STV_Geography(g))))) FROM geogs;
ST_AsText
---------------------
POINT (-98.424499 44.05034775)
(1 row)
```

STV_GeographyPoint

Returns a GEOGRAPHY point based on the input values.

This is the optimal way to convert raw coordinates to GEOGRAPHY points.
Behavior Type

Immutable

Syntax

STV_GeographyPoint( x, y )

Arguments

<table>
<thead>
<tr>
<th>x</th>
<th>x-coordinate or longitude, FLOAT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>y-coordinate or latitude, FLOAT.</td>
</tr>
</tbody>
</table>

Returns

GEOGRAPHY

Examples

The following examples show how to use STV_GeographyPoint.

Return a GEOGRAPHY point:

```sql
=> SELECT ST_AsText(STV_GeographyPoint(-114.101588, 47.909677));
   ST_AsText
  -------------------------------------------
 POINT (-114.101588 47.909677)
 (1 row)
```

Return GEOGRAPHY points using two columns:

```sql
=> CREATE TABLE geog_data (id IDENTITY, x FLOAT, y FLOAT);
CREATE TABLE
=> COPY geog_data FROM STDIN;
 Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
>> -114.181588|47.909677
>> -111.532377|46.430753
>> 
=> SELECT id, ST_AsText(STV_GeographyPoint(x, y)) FROM geog_data;
   id | ST_AsText
   ----|-------------------
1 | POINT (-114.101588 47.909677)
2 | POINT (-111.532377 46.430753)
```
Create GEOGRAPHY points by manipulating data source columns during load:

```sql
> CREATE TABLE geog_data_load (id IDENTITY, geog GEOGRAPHY);
> CREATE TABLE
> COPY geog_data_load (lon FILLER FLOAT,
> lat FILLER FLOAT,
> geog AS STV_GeographyPoint(lon, lat))
> FROM 'test_coords.csv' DELIMITER ',';
> Rows Loaded
> 2
> (1 row)
> > SELECT id, ST_AsText(geog) FROM geog_data_load;
> id | ST_AsText
> --------
> 1 | POINT (-114.101588 47.909677)
> 2 | POINT (-111.532377 46.430753)
> (2 rows)
```

See Also

STV_GeometryPoint

STV_Geometry

Casts a GEOGRAPHY object into a GEOMETRY object.
The SRID value does not affect the results of Vertica Place queries.

Behavior Type

Immutable

Syntax

```sql
STV_Geometry( geog )
```
Arguments

| geog         | Spatial object that you want to cast into a GEOMETRY object, type GEOGRAPHY |

Returns

GEOMETRY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use STV_Geometry.

Convert the GEOGRAPHY values to GEOMETRY values, then convert the result back to a GEOGRAPHY type:

```verbatim
=> CREATE TABLE geogs(g GEOGRAPHY);
=> CREATE TABLE
=> COPY geogs(gx filler LONG VARCHAR, geog AS ST_GeographyFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> MULTIPOINT(-108.619726 45.000284,-107.866813 45.00107,-106.363711 44.994223,-70.847746 41.205814)
=> \.
=> SELECT ST_AsText(ST_Geography(ST_Centroid(STV_Geometry(g)))) FROM geogs;
ST_AsText
```
### STV_GeometryPoint

Returns a GEOMETRY point, based on the input values.

This approach is the most-optimal way to convert raw coordinates to GEOMETRY points.

#### Behavior Type

Immutable

#### Syntax

```
STV_GeometryPoint( x, y [, srid] )
```

#### Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x-coordinate or longitude, FLOAT.</td>
</tr>
<tr>
<td>y</td>
<td>y-coordinate or latitude, FLOAT.</td>
</tr>
<tr>
<td>srid</td>
<td>(Optional) Spatial Reference Identifier (SRID) assigned to the point, INTEGER.</td>
</tr>
</tbody>
</table>

#### Returns

GEOMETRY

#### Examples

The following examples show how to use STV_GeometryPoint.

Return a GEOMETRY point with an SRID:

```sql
=> SELECT ST_AsText(STV_GeometryPoint(71.148562, 42.989374, 4326));
ST_AsText
-------------------
POINT (71.148562, 42.989374)
```
POINT (-71.148562 42.989374)
(1 row)

Return GEOMETRY points using two columns:

```sql
=> CREATE TABLE geom_data (id IDENTITY, x FLOAT, y FLOAT, SRID INT);
=> COPY geom_data FROM STDIN;
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
=> 42.36383053600048|-71.10165445099966|4326
=> 42.36789379800085|-71.10644448699964|4326
=> 
=> SELECT id, ST_AsText(STV_GeometryPoint(x, y, SRID)) FROM geom_data;
```

<table>
<thead>
<tr>
<th>id</th>
<th>ST_AsText</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (-71.101654 42.36383)</td>
</tr>
<tr>
<td>2</td>
<td>POINT (-71.106444 42.36789)</td>
</tr>
</tbody>
</table>

(2 rows)

Create GEOMETRY points by manipulating data source columns during load:

```sql
=> CREATE TABLE geom_data_load (id IDENTITY, geom GEOMETRY);
CREATE TABLE
=> COPY geom_data_load (lon FILLER FLOAT,
                        lat FILLER FLOAT,
                        geom AS STV_GeometryPoint(lon, lat))
                        FROM 'test_coords.csv' DELIMITER ',';
Rows Loaded
-------------
2
(1 row)
=> SELECT id, STAsText(geom) FROM geom_data_load;
```

<table>
<thead>
<tr>
<th>id</th>
<th>STAsText</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POINT (-75.101654 43.36383)</td>
</tr>
<tr>
<td>2</td>
<td>POINT (-75.106444 43.36789)</td>
</tr>
</tbody>
</table>

(2 rows)

See Also

STV_GeographyPoint

STV_GetExportShapefileDirectory

Returns the path of the export directory.
Behavior Type

Immutable

Syntax

```
STV_GetExportShapefileDirectory()
```

Returns

The path of the shapefile export directory.

Examples

The following example shows how you can use STV_GetExportShapefileDirectory to query the path of the shapefile export directory:

```
=> SELECT STV_GetExportShapefileDirectory();
  STV_GetExportShapefileDirectory
  -------------------------------------
  Shapefile export directory: [/home/user/temp]
  (1 row)
```

STV_Intersect Scalar Function

Spatially intersects a point or points with a set of polygons. The STV_Intersect scalar function returns the identifier associated with an intersecting polygon.

Behavior Type

Immutable

Syntax

```
STV_Intersect( { g | x, y } 
  USING PARAMETERS index='index_name')
```
**Arguments**

<table>
<thead>
<tr>
<th>g</th>
<th>A geometry or geography (WGS84) column that contains points. The g column can contain only point geometries or geographies. If the column contains a different geometry or geography type, STV_Intersect terminates with an error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x-coordinate or longitude, FLOAT.</td>
</tr>
<tr>
<td>y</td>
<td>y-coordinate or latitude, FLOAT.</td>
</tr>
</tbody>
</table>

**Parameters**

| index = 'index_name' | Name of the spatial index, of type VARCHAR. |

**Returns**

The identifier of a matching polygon. If the point does not intersect any of the index's polygons, then the STV_Intersect scalar function returns NULL.

**Examples**

The following examples show how you can use STV_Intersect scalar.

Using two floats, return the gid of a matching polygon or NULL:

```sql
=> CREATE TABLE pols (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY pols(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> 1|POLYGON((31 74,8 70,8 50,36 53,31 74))
=> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true, max_mem_mb=256) OVER() FROM pols;
  type | polygons | SRID | min_x | min_y | max_x | max_y | info
-------|----------|------|-------|-------|-------|-------|-------
GEOMETRY | 1 | 0 | 8 | 50 | 36 | 74 |
(1 row)

=> SELECT STV_Intersect(12.5683, 55.6761 USING PARAMETERS index = 'my_polygons_1');
STV_Intersect
```

Vertica Analytic Database (9.0.x) Page 2607 of 6180
Using a GEOMETRY column, return the gid of a matching polygon or NULL:

```sql
=> CREATE TABLE polygons (gid INT, geom GEOMETRY(700));
CREATE TABLE
=> COPY polygons (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.

| POLYGON((31 74,8 70,8 50,-36 53,-31 74))
| POLYGON((-38 50,4 13,11 45,0 65,-38 50))
| POLYGON((-18 42,-10 65,27 48,14 26,-18 42))
=> .
=> SELECT ST_Create_Index(gid, geom USING PARAMETERS index='my_polygons', overwrite=true,
                     max_mem_mb=256) OVER() FROM polygons;

| GEOMETRY | polygons | SRID | min_x | min_y | max_x | max_y | info
----------|----------|------|-------|-------|-------|-------|-------
| GEOMETRY | 3        | 0    | -38   | 13    | 27    | 74    |       
(1 row)

=> CREATE TABLE points (gid INT, geom GEOMETRY(700));
CREATE TABLE
=> COPY points (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.

| POINT(-1 52)
| POINT(-20 0)
| POINT(-8 25)
| POINT(0 0)
| POINT(1 5)
| POINT(20 45)
| POINT(-20 5)
| POINT(-20 1)
=> .
=> SELECT pt_gid AS gid, STV_Intersect(geom USING PARAMETERS index='my_polygons') AS pol_gid
    FROM points ORDER BY pt_gid;

| pt_gid | pol_gid
----------|--------
| 100    | 1      
| 101    | 1      
| 102    | 2      
| 103    | 2      
| 104    | 3      
| 105    | 3      
| 106    | 3      
| 107    | 3      
(8 rows)
```

See Also

- Best Practices for Spatial Joins
- STV_Intersect: Scalar Function vs. Transform Function
• STV_Intersect Transform Function

• STV_Create_Index

**STV_Intersect Transform Function**

Spatially intersects points and polygons. The STV_Intersect transform function returns a tuple with matching point/polygon pairs. For every point, Vertica returns either one or many matching polygons.

You can improve performance when you parallelize the computation of the STV_Intersect transform function over multiple nodes. To parallelize the computation, use an OVER (PARTITION BEST) clause.

**Behavior Type**

Immutable

**Syntax**

```
STV_Intersect ( { gid | i }, { g | x, y } )
   USING PARAMETERS index='index_name'
   OVER() AS (pt_gid, pol_gid)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**gid</td>
<td>i**</td>
</tr>
<tr>
<td><strong>g</strong></td>
<td>A geometry or geography (WGS84) column that contains points. The g column can contain only point geometries or geographies. If the column contains a different geometry or geography type, STV_Intersect terminates with an error.</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>x-coordinate or longitude, FLOAT.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>y-coordinate or latitude, FLOAT.</td>
</tr>
</tbody>
</table>
Parameters

| index = 'index_name' | Name of the spatial index, of type VARCHAR. |

Returns

<table>
<thead>
<tr>
<th>pt_gid</th>
<th>Unique identifier of the point geometry or geography, of type INTEGER.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol_gid</td>
<td>Unique identifier of the polygon geometry or geography, of type INTEGER.</td>
</tr>
</tbody>
</table>

Examples

The following examples show how you can use STV_Intersect transform.

Using two floats, return the matching point-polygon pairs.

```sql
=> CREATE TABLE pols (gid INT, geom GEOMETRY(1000));
CREATE TABLE
=> COPY pols(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> 1|POLYGON((31 74,8 70,8 50,36 53,31 74))
=> 
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true, max_mem_mb=256) OVER() FROM pols;
```

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>50</td>
<td>36</td>
<td>74</td>
<td>(1 row)</td>
</tr>
</tbody>
</table>

=> SELECT STV_Intersect(56, 12.5683, 55.6761 USING PARAMETERS index = 'my_polygons_1') OVER();
```
```
<table>
<thead>
<tr>
<th>pt_gid</th>
<th>pol_gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>1</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>

Using a GEOMETRY column, return the matching point-polygon pairs.

```sql
=> CREATE TABLE polygons (gid int, geom GEOMETRY(700));
CREATE TABLE
=> COPY polygons (gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
=> 10|POLYGON((5 5, 5 10, 10 10, 10 5, 5 5))
=> 11|POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))
=> 12|POLYGON((1 1, 1 3, 3 3, 3 1, 1 1))
```
```sql```
You can improve query performance by using the STV_Intersect transform function in a WHERE clause. Performance improves because this syntax eliminates all points that do not intersect polygons in the index.

Return the count of points that intersect with the polygon, where gid = 14:

```sql
=> SELECT COUNT(pt_id) FROM
(SELECT STV_Intersect(gid, geom USING PARAMETERS index='my_polygons') OVER (PARTITION BEST) AS (point_id, polygon_gid)
FROM points)
AS T WHERE pol_id = 14;

<table>
<thead>
<tr>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
```
See Also

- Best Practices for Spatial Joins
- STV_Intersect: Scalar Function vs. Transform Function
- STV_Create_Index
- STV_Intersect Scalar Function

**STV_IsValidReason**

Determines if a spatial object is well formed or valid. If the object is not valid, STV_IsValidReason returns a string that explains where the invalidity occurs.

A polygon or multipolygon is valid if all of the following are true:

- The polygon is closed; its start point is the same as its end point.
- Its boundary is a set of linestrings.
- The boundary does not touch or cross itself.
- Any polygons in the interior that do not have more than one point touching the boundary of the exterior polygon.

If you pass an invalid object to a Vertica Place function, the function fails or returns incorrect results. To determine if a polygon is valid, first run ST_IsValid. ST_IsValid returns TRUE if the polygon is valid, FALSE otherwise.

Important: STV_IsValidReason supports only polygon and multipolygon GEOMETRY data types.

**Behavior Type**

Immutable

**Syntax**

STV_IsValidReason( \( g \) )
Arguments

| g | Geospatial object to test for validity, value of type GEOMETRY or GEOGRAPHY (WGS84). |

Returns

LONG VARCHAR

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use STV_IsValidReason.

Returns a string describing where the polygon is invalid:

```sql
=> SELECT STV_IsValidReason(ST_GeomFromText('POLYGON((1 3,3 2,1 1, 3 0,1 0,1 3))'));
STV_IsValidReason
-----------------------------
Ring Self-intersection at or near POINT (1 1)
(1 row)
```
See Also

ST_IsValid

STV_LineStringPoint

Retrieves the vertices of a linestring or multilinestring. The values returned are points of either GEOMETRY or GEOGRAPHY type depending on the input object's type. GEOMETRY points inherit the SRID of the input object.

STV_LineStringPoint is an analytic function. For more information, see Analytic Functions.

Behavior Type

Immutable

Syntax

\[
\text{STV\_LineStringPoint}( g ) \quad \text{OVER( [PARTITION NODES] ) AS}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g )</td>
<td>Linestring or multilinestring, value of type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

Returns

GEOMETRY or GEOGRAPHY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
</table>
Examples

The following examples show how to use STV_LineStringPoint.

Returns the vertices of the geometry linestring and their SRID:

```sql
=> SELECT ST_Astext(Point), ST_SRID(Point)
FROM (SELECT STV_LineStringPoint(
    ST_GeomFromText('MULTILINESTRING((1 2, 2 3, 3 1, 4 2),
    (10 20, 20 30, 30 10, 40 20))', 4269)) OVER () AS Point) AS foo;

<table>
<thead>
<tr>
<th>ST_Astext</th>
<th>ST_SRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT (1 2)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (2 3)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (3 1)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (4 2)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (10 20)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (20 30)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (30 10)</td>
<td>4269</td>
</tr>
<tr>
<td>POINT (40 20)</td>
<td>4269</td>
</tr>
</tbody>
</table>
(8 rows)
```

Returns the vertices of the geography linestring:

```sql
=> SELECT ST_Astext(g)
FROM (SELECT STV_LineStringPoint(
    ST_GeographyFromText('MULTILINESTRING ((42.1 71.0, 41.4 70.0, 41.3 72.9),
    (42.99 71.46, 44.47 73.21))', 4269)) OVER () AS g) AS line_geog_points;

<table>
<thead>
<tr>
<th>ST_Astext</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT (42.1 71.0)</td>
</tr>
<tr>
<td>POINT (41.4 70.0)</td>
</tr>
<tr>
<td>POINT (41.3 72.9)</td>
</tr>
<tr>
<td>POINT (42.99 71.46)</td>
</tr>
<tr>
<td>POINT (44.47 73.21)</td>
</tr>
</tbody>
</table>
(5 rows)
```
See Also

STV_PolygonPoint

**STV_MemSize**

Returns the length of the spatial object in bytes as an INTEGER.
Use this function to determine the optimal column width for your spatial data.

**Behavior Type**

Immutable

**Syntax**

```
STV_MemSize( g )
```

**Arguments**

<table>
<thead>
<tr>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial object, value of type GEOMETRY or GEOGRAPHY</td>
</tr>
</tbody>
</table>

**Returns**

INTEGER

**Examples**

The following example shows how you can optimize your table by sizing the GEOMETRY or GEOGRAPHY column to the maximum value returned by STV_MemSize:

```
=> CREATE TABLE mem_size_table (id int, geom geometry(800));
CREATE TABLE
=> COPY mem_size_table (id, gx filler LONG VARCHAR, geom as ST_GeomFromText(gx)) FROM STDIN DELIMITER ' |
Enter data to be copied followed by a newline.
```
End with a backslash and a period on a line by itself.

```sql
>\.
=> SELECT max(STV_MemSize(geom)) FROM mem_size_table;
     max
    -----
     336
(1 row)
```

```sql
=> CREATE TABLE production_table(id int, geom geometry(336));
CREATE TABLE
=> INSERT INTO production_table SELECT * FROM mem_size_table;
CREATE TABLE
=> DROP mem_size_table;
DROP TABLE
```

**STV_NN**

Calculates the distance of spatial objects from a reference object and returns (object, distance) pairs in ascending order by distance from the reference object.

Parameters $g_1$ and $g_2$ must be both GEOMETRY objects or both GEOGRAPHY objects.

**STV_NN** is an analytic function. For more information, see Analytic Functions.

**Behavior Type**

Immutable

**Syntax**

```sql
STV_NN( g, ref_obj, k ) OVER()
```

**Arguments**

<table>
<thead>
<tr>
<th>$g$</th>
<th>Spatial object, value of type GEOMETRY or GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ref_obj$</td>
<td>Reference object, type GEOMETRY or GEOGRAPHY</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of rows to return, type INTEGER</td>
</tr>
</tbody>
</table>
Returns

(Object, distance) pairs, in ascending order by distance. If a parameter is EMPTY or NULL, then 0 rows are returned.

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipoint</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use STV_NN.

Create a table and insert nine GEOGRAPHY points:

```sql
=> CREATE TABLE points (g geography);
CREATE TABLE
=> COPY points (gx filler LONG VARCHAR, g AS ST_GeographyFromText(gx)) FROM stdin delimiter ' | ';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> POINT (21.5 18.4)  
>> POINT (21.5 19.2)  
>> POINT (21.5 20.7)  
>> POINT (22.5 16.4)  
>> POINT (22.5 17.15) 
>> POINT (22.5 18.33) 
>> POINT (23.5 13.68) 
>> POINT (23.5 15.9)  
>> POINT (23.5 18.4)  
>> .
```
Calculate the distances (in meters) of objects in table points from the GEOGRAPHY point (23.5, 20).

Returns the five objects that are closest to that point:

```sql
=> SELECT ST_AsText(nn), dist FROM (SELECT STV_PolygonPoint(g, ST_GeographyFromText('POINT(23.5 20)'),5) OVER() AS (nn, dist) FROM points) AS example;
```

<table>
<thead>
<tr>
<th>ST_AsText</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT (23.5 18.4)</td>
<td>177912.12757541</td>
</tr>
<tr>
<td>POINT (22.5 18.33)</td>
<td>213339.210738322</td>
</tr>
<tr>
<td>POINT (21.5 20.7)</td>
<td>222561.43679943</td>
</tr>
<tr>
<td>POINT (21.5 19.2)</td>
<td>227604.371833335</td>
</tr>
<tr>
<td>POINT (21.5 18.4)</td>
<td>275239.416790128</td>
</tr>
</tbody>
</table>

(5 rows)

**STV_PolygonPoint**

Retrieves the vertices of a polygon as individual points. The values returned are points of either GEOMETRY or GEOGRAPHY type depending on the input object’s type. GEOMETRY points inherit the SRID of the input object.

STV_PolygonPoint is an analytic function. For more information, see Analytic Functions.

**Behavior Type**

Immutable

**Syntax**

```sql
STV_PolygonPoint( g )
OVER( [PARTITION NODES] ) AS
```

**Arguments**

| g   | Polygon, value of type GEOMETRY or GEOGRAPHY |

**Returns**

GEOMETRY or GEOGRAPHY
## Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

## Examples

The following examples show how to use STV_PolygonPoint.

**Returns the vertices of the geometry polygon:**

```sql
=> SELECT ST_AsText(g) FROM (SELECT STV_PolygonPoint(ST_GeomFromText('POLYGON((1 2, 2 3, 3 1, 1 2)'))
    OVER (PARTITION NODES) AS g) AS poly_points;

ST_AsText
---------------------
POINT (1 2)
POINT (2 3)
POINT (3 1)
POINT (1 2)
(4 rows)
```

**Returns the vertices of the geography polygon:**

```sql
=> SELECT ST_AsText(g) FROM (SELECT STV_PolygonPoint(ST_GeographyFromText('POLYGON((25.5 28.76, 28.83 29.13, 27.2 30.99, 25.5 28.76))'))
    OVER (PARTITION NODES) AS g) AS poly_points;

ST_AsText
---------------------
POINT (25.5 28.76)
POINT (28.83 29.13)
POINT (27.2 30.99)
POINT (25.5 28.76)
```
See Also

STV_LineStringPoint

STV_Reverse
Reverses the order of the vertices of a spatial object.

Behavior Type
Immutable

Syntax

STV_Reverse( g, [USING PARAMETERS skip_nonreorientable_polygons={true | false} ])

Arguments

<table>
<thead>
<tr>
<th>g</th>
<th>Spatial object, type GEOGRAPHY.</th>
</tr>
</thead>
</table>
| skip_nonreorientable_polygons = { true | false } | (Optional) Boolean
When set to False, non-orientable polygons generate an error. For example, if you use STV_ForceLHR or STV_Reverse with skip_nonorientable_polygons set to False, a geography polygon containing a hole generates an error. When set to True, the result returned is the polygon, as passed to the API, without alteration.

This argument can help you when you are creating an index from a table containing polygons that cannot be re-oriented.

Vertica Place considers these polygons non-
orientable:
- Polygons with a hole
- Multipolygons
- Multipolygons with a hole

Default value: False

Returns

GEOGRAPHY

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (Perfect Sphere)</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples

The following examples show how you can use STV_Reverse.

Reverse vertices of a geography polygon:

```sql
=> SELECT ST_AsText(STV_Reverse(ST_GeographyFromText('Polygon((1 1, 3 1, 2 2, 1 1))')));
ST_AsText
----------------------
```

```sql
----------------------
```
Force the polygon to reverse orientation:

```sql
=> SELECT ST_AsText(ST_Reverse(ST_GeographyFromText('Polygon((1 1, 2 2, 3 1, 1 1))')));
POLYGON ((1 1, 3 1, 2 2, 1 1))
(1 row)
```

See Also

*STV_ForceLHR*

**STV_Rename_Index**

Renames a spatial index. If the index format is out of date, you cannot rename the index.

A spatial index is created from an input polygon set, which can be the result of a query. Spatial indexes are created in a global name space. Vertica uses a distributed plan whenever the input table or projection is segmented across nodes of the cluster.

The OVER() clause must be empty.

**Behavior Type**

Immutable

**Syntax**

```sql
STV_Rename_Index( USING PARAMETERS
    source = 'old_index_name',
    dest = 'new_index_name',
    overwrite = [ 'true' | 'false' ]
) OVER ()
```

**Arguments**

| source = 'old_index_name' | Current name of the spatial index, type VARCHAR. |
**dest = 'new_index_name'**  
New name of the spatial index, type VARCHAR.

**overwrite = [ 'true' | 'false' ]**  
(Optional) BOOLEAN value that specifies whether to overwrite the index, if an index exists. This parameter cannot be NULL.  
Default: False

### Privileges

Any user with access to the STV_*_Index functions can describe, rename, or drop indexes created by any other user.

### Limitations

- Index names cannot exceed 110 characters.
- The backslash or tab characters are not allowed in index names.

### Example

The following example shows how to use STV_Rename_Index.

**Rename an index:**

```sql
=> SELECT STV_Rename_Index (  
    USING PARAMETERS  
    source = 'my_polygons',  
    dest = 'US_states',  
    overwrite = 'false'  
)  
OVER ();  
rename_index  
-------------  
Index renamed  
(1 Row)
```

### STV_Refresh_Index

Appends newly added or updated polygons and removes deleted polygons from an existing spatial index.

The OVER() clause must be empty.
Behavior Type
Mutable

Syntax

STV_Refresh_Index( gid, g
    USING PARAMETERS index='index_name'
    [, skip_nonindexable_polygons={ true | false } ]
)
OVER()

[ AS (type, polygons, srid, min_x, min_y, max_x, max_y, info,
    indexed, appended, updated, deleted) ]

Arguments

<table>
<thead>
<tr>
<th>gid</th>
<th>Name of an integer column that uniquely identifies the polygon. The gid cannot be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Name of a geometry or geography (WGS84) column or expression that contains polygons and multipolygons. Only polygon and multipolygon can be indexed. Other shape types are excluded from the index.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>index = 'index_name'</th>
<th>Name of the index, type VARCHAR. Index names cannot exceed 110 characters. The slash, backslash, and tab characters are not allowed in index names.</th>
</tr>
</thead>
</table>
| skip_nonindexable_polygons = { true | false } | (Optional) BOOLEAN
In rare cases, intricate polygons (for instance, with too high resolution or anomalous spikes) cannot be indexed. These polygons are considered non-indexable. When set to False, non-indexable polygons cause the index creation to fail. When set to True, index creation can succeed by excluding non-indexable polygons from the index. |
To review the polygons that were not able to be indexed, use STV_Describe_Index with the parameter list_polygon.
Default: False

Returns

<table>
<thead>
<tr>
<th>type</th>
<th>Spatial object type of the index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygons</td>
<td>Number of polygons indexed.</td>
</tr>
<tr>
<td>SRID</td>
<td>Spatial reference system identifier.</td>
</tr>
<tr>
<td>min_x, min_y, max_x, max_y</td>
<td>Coordinates of the minimum bounding rectangle (MBR) of the indexed geometries. (min_x, min_y) are the south-west coordinates, and (max_x, max_y) are the north-east coordinates.</td>
</tr>
<tr>
<td>info</td>
<td>Lists the number of excluded spatial objects as well as their type that were excluded from the index.</td>
</tr>
<tr>
<td>indexed</td>
<td>Number of polygons indexed during the operation.</td>
</tr>
<tr>
<td>appended</td>
<td>Number of appended polygons.</td>
</tr>
<tr>
<td>updated</td>
<td>Number of updated polygons.</td>
</tr>
<tr>
<td>deleted</td>
<td>Number of deleted polygons.</td>
</tr>
</tbody>
</table>

Supported Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>GEOMETRY</th>
<th>GEOGRAPHY (WGS84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multipoint</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linestring</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multilinestring</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Multipolygon</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Privileges**

Any user with access to the STV_*_*Index functions can describe, rename, or drop indexes created by any other user.

**Limitations**

- In rare cases, intricate polygons (such as those with too-high a resolution or anomalous spikes) cannot be indexed. See the parameter skip_nonindexable_polygons.

- If you replace a valid polygon in the source table with an invalid polygon, STV_Refresh_ Index ignores the invalid polygon. As a result, the polygon originally indexed persists in the index.

- The following geometries cannot be indexed:
  - Non-polygons
  - NULL gid
  - NULL (multi) polygon
  - EMPTY (multi) polygon
  - Invalid (multi) polygon

- The following geographies are excluded from the index:
  - Polygons with holes
  - Polygons crossing the International Date Line
  - Polygons covering the north or south pole
  - Antipodal polygons
Usage Tips

- To cancel an STV_Refresh_Index run, use Ctrl + C.

- If you use source data not previously associated with the index, then the index will be overwritten.

- If STV_Refresh_Index has insufficient memory to process the query, then rebuild the index using STV_Create_Index.

- If there are no valid polygons in the geom column, STV_Refresh_Index reports an error in vertica.log and stops the index refresh.

- Ensure that all of the polygons you plan to index are valid polygons. STV_Create_Index and STV_Refresh_Index do not check polygon validity when building an index.

For more information, see Ensuring Polygon Validity Before Creating or Refreshing an Index.

Examples

The following examples show how to use STV_Refresh_Index.

Refresh an index with a single literal argument:

```sql
=> SELECT STV_Create_Index(1, ST_GeomFromText('POLYGON((0 0,0 15.2,3.9 15.2,3.9 0,0 0))')
       USING PARAMETERS index='my_polygon') OVER();

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.9</td>
<td>15.2</td>
<td></td>
</tr>
</tbody>
</table>
(1 row)
```

```sql
=> SELECT STV_Refresh_Index(2, ST_GeomFromText('POLYGON((0 0,0 15.2,3.9 15.2,3.9 0,0 0))')
       USING PARAMETERS index='my_polygon') OVER();

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
<th>info</th>
<th>indexed</th>
<th>appended</th>
<th>updated</th>
<th>deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.9</td>
<td>18.2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
(1 row)
```

Refresh an index from a table:

```sql
=> CREATE TABLE polys (gid INT, geom GEOMETRY);
CREATE TABLE
=> COPY polys(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
Enter data to be copied followed by a newline.
```
End with a backslash and a period on a line by itself.

```sql
=> 1|POLYGON((31 74, 70, 8 50, -36 53, -31 74))
=> 2|POLYGON((5 20, 9 30, 20 45, 36 35, 5 20))
=> 3|POLYGON((12 23, 9 30, 20 45, 36 35, 37 67, 45 80, 50 20, 12 23))
=> \.
=> SELECT STV_Create_Index(gid, geom USING PARAMETERS index='my_polygons_1', overwrite=true)
   OVER() FROM pols;
```

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
<th>info</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>3</td>
<td>0</td>
<td>-36</td>
<td>20</td>
<td>50</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

```sql
=> COPY pols(gid, gx filler LONG VARCHAR, geom AS ST_GeomFromText(gx)) FROM stdin delimiter '|';
```

Enter data to be copied followed by a newline.

```sql
=> 6|POLYGON((-32 74, 70, 8 50, -36 53, -32 74))
=> \.
=> SELECT STV_Refresh_Index(gid, geom USING PARAMETERS index='my_polygons_1') OVER() FROM pols;
```

<table>
<thead>
<tr>
<th>type</th>
<th>polygons</th>
<th>SRID</th>
<th>min_x</th>
<th>min_y</th>
<th>max_x</th>
<th>max_y</th>
<th>info</th>
<th>indexed</th>
<th>appended</th>
<th>updated</th>
<th>deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEOMETRY</td>
<td>4</td>
<td>0</td>
<td>-36</td>
<td>20</td>
<td>50</td>
<td>80</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

See Also

- STV_Create_Index
- STV_Describe_Index
- STV_Drop_Index
- STV_Rename_Index
- Ensuring Polygon Validity Before Creating or Refreshing an Index

**STV_SetExportShapefileDirectory**

Specifies the directory to export GEOMETRY or GEOGRAPHY data to a shapefile. The validity of the path is not checked, and the path cannot be empty.

**Behavior Type**

Immutable
Syntax

`STV_SetExportShapefileDirectory( USING PARAMETERS path='shapefile_path' )`

Arguments

| path = 'shapefile_path' | The path where you want the shapefile exported. For example, '/home/user/temp'. |

Returns

The path of the shapefile export directory.

Privileges

Only a superuser can use this function.

Examples

The following example shows how you can use `STV_SetExportShapefileDirectory` to set the shapefile export directory to /home/user/temp:

```sql
=> SELECT STV_SetExportShapefileDirectory(USING PARAMETERS path = '/home/user/temp');
```

SUCCESS. Set shapefile export directory: [/home/user/temp]  
(1 row)

**STV_ShpSource and STV_ShpParser**

These two functions work with a COPY command to parse and load the geometries and attributes from a shapefile into a database table and convert them to the GEOMETRY data type format. You must use these two functions together. An SRID is required. An empty multipoint or an invalid multipolygon can not be loaded from a shapefile.
Behavior Type

Immutable

Syntax

COPY table_name( col2, col3, ..., coln )
    WITH SOURCE STV_ShpSource( file = 'filename'[, SRID=spatial reference identifier]
        [, flatten_2d={true | false}] ] )
    PARSER STV_ShpParser()

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table in which to load the geometry data.</td>
</tr>
<tr>
<td>col1, col2, ...</td>
<td>Column names in the table that match the fields in the external file. Run the CREATE TABLE command that STV_ShpCreateTable creates. When you do so, these columns correspond to the second through the second-to-last columns.</td>
</tr>
<tr>
<td>file = 'filename'</td>
<td>Fully qualified path of the .dbf, .shp, or .shx file.</td>
</tr>
<tr>
<td>SRID=spatial reference identifier</td>
<td>Spatial reference identifier (SRID) associated with the shape file, type INTEGER.</td>
</tr>
<tr>
<td>flatten_2d</td>
<td>(Optional) BOOLEAN that excludes 3D or 4D coordinates during COPY commands.</td>
</tr>
<tr>
<td></td>
<td>True - Excludes geometries with 3D or 4D coordinates before a COPY command.</td>
</tr>
<tr>
<td></td>
<td>False - Causes the load to fail if a geometry with 3D or 4D coordinate is found.</td>
</tr>
<tr>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>
Usage Tips

- The COPY command fails if:
  - The shapefile cannot be located or opened.
  - The number of columns or the data types of the columns that STV_ShpParser creates do not match the columns in the destination table. Use STV_ShpCreateTable to generate the appropriate CREATE TABLE command.
  - One of the mandatory files is missing or cannot be opened. When opening a shapefile, you must have three files: .dbf, .shp, and .shx.
  - If the .shp and .shx files are corrupt, STV_ShpSource returns an error. If the .shp and .shx files are valid, but the .dbf file is corrupt, STV_ShpSource ignores the .dbf file and does not create columns for that data.
  - Any rejected records are saved in the /CopyErrorLogs directory under the catalog directory.
  - If the .dbf component of a shapefile contains a numeric attribute, this field's values may lose precision when the Vertica Place shapefile loader loads it into a table. The target field is a 64-bit FLOAT column, which can only represent about 15 significant digits; in a .dbf file, Numeric fields can be up to 30 digits.

Example

The following example shows how to use STV_ShpSource and STV_ShpParser.

```
=> COPY tl_2010_us_state10 WITH SOURCE STV_ShpSource(file='/shapefiles/tl_2010_us_state10.shp', SRID=4269) PARSER STV_ShpParser();
Rows loaded
----------
 52
```

See Also

- STV_ShpCreateTable
STV_ShpCreateTable

Returns a CREATE TABLE statement with the columns and types of the attributes found in the specified shapefile.

The column types are sized according to the shapefile metadata. The size of the column is based on the largest geometry found in the shapefile. The first column in the table is gid, which is an auto-increment IDENTITY primary key column. The cache value is set to 64 by default. The last column is a GEOMETRY data type for storing the actual geometry data.

Behavior Type

Immutable

Syntax

STV_ShpCreateTable(USING PARAMETERS file='filename') OVER()

Arguments

| file = 'filename' | Fully qualified path of the .dbf, .shp, or .shx file. (The extension is optional.) |

Returns

CREATE TABLE statement that matches the specified shapefile

Usage Tips

- STV_ShpCreateTable returns a CREATE TABLE statement; but it does not create the table. Modify the CREATE TABLE statement as needed, and then create the table before loading the shapefile into the table.

- To create a table with characters other than alphanumeric and underscore (_) characters, you must specify the table name enclosed in double quotes, such as "counties%NY".
The name of the table is the same as the name of the shapefile, without the directory name or extension.

The shapefile must be accessible from the initiator node.

If the `.shp` and `.shx` files are corrupt, STV_ShpCreateTable returns an error. If the `.shp` and `.shx` files are valid, but the `.dbf` file is corrupt, STV_ShpCreateTable ignores the `.dbf` file and does not create columns for that data.

All the mandatory files (.dbf, .shp, .shx) must be in the same directory. If not, STV_ShpCreateTable returns an error.

If the `.dbf` component of a shapefile contains a Numeric attribute, this field's values may lose precision when the Vertica shapefile loader loads it into a table. The target field is a 64-bit FLOAT column, which can only represent about 15 significant digits. In a `.dbf` file, numeric fields can be up to 30 digits.

Vertica records all instances of shapefile values that are too long in the vertica.log file.

Example

The following example shows how to use STV_ShpCreateTable.

Returns a CREATE TABLE statement:

```sql
=> SELECT STV_ShpCreateTable
    (USING PARAMETERS file='/shapefiles/tl_2010_us_state10.shp')
  OVER() as create_table_states;
create_table_states
------------------------------------------
CREATE TABLE tl_2010_us_state10(
    gid IDENTITY(64) PRIMARY KEY,
    REGION10 VARCHAR(2),
    DIVISION10 VARCHAR(2),
    STATEFP10 VARCHAR(2),
    STATENS10 VARCHAR(8),
    GEOID10 VARCHAR(2),
    STUSPS10 VARCHAR(2),
    NAME10 VARCHAR(100),
    LSAD10 VARCHAR(2),
    MTFC10 VARCHAR(5),
    FUNCSTAT10 VARCHAR(1),
    ALAND10 INT8,
    AWATER10 INT8,
    INTPTLAT10 VARCHAR(11),
    INTPTLON10 VARCHAR(12),
    geom GEOMETRY(940845)
);
(18 rows)
```
See Also

- `STV_ShpSource` and `STV_ShpParser`
IP Conversion Functions

IP functions perform conversion, calculation, and manipulation operations on IP, network, and subnet addresses.

INET_ATON

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

Behavior Type

Immutable

Syntax

INET_ATON ( expression )

Parameters

expression (VARCHAR) is the string to convert.

Notes

The following syntax converts an IPv4 address represented as the string A to an integer I. INET_ATON trims any spaces from the right of A, calls the Linux function inet_pton, and converts the result from network byte order to host byte order using ntohs.

=> INET_ATON(VARCHAR A) -> INT8 I

If A is NULL, too long, or inet_pton returns an error, the result is NULL.
Examples

The generated number is always in host byte order. In the following example, the number is calculated as 209×256^3 + 207×256^2 + 224×256 + 40.

```sql
=> SELECT INET_ATON('209.207.224.40');
  inet_aton
-------------
3520061480
(1 row)

=> SELECT INET_ATON('1.2.3.4');
  inet_aton
-------------
16909060
(1 row)

=> SELECT TO_HEX(INET_ATON('1.2.3.4'));
  to_hex
-------------
1020304
(1 row)
```

See Also

- INET_NTOA

INET_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.

Behavior Type

Immutable

Syntax

INET_NTOA ( expression )
Parameters

expression (INTEGER) is the network address to convert.

Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET_NTOA converts I from host byte order to network byte order using htonl, and calls the Linux function inet_ntop.

=> INET_NTOA(INT8 I) -> VARCHAR A

If I is NULL, greater than $2^{32}$ or negative, the result is NULL.

Examples

=> SELECT INET_NTOA(16909060);
   inet_ntoa
   ---------------
   1.2.3.4
   (1 row)

=> SELECT INET_NTOA(03021962);
   inet_ntoa
   ---------------
   0.46.28.138
   (1 row)

See Also

- INET_ATON

V6_ATON

Converts an IPv6 address represented as a character string to a binary string.
Behavior Type

Immutable

Syntax

V6_ATON ( expression )

Parameters

expression (VARCHAR) is the string to convert.

Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6_ATON trims any spaces from the right of A and calls the Linux function inet_pton.

=> V6_ATON(VARCHAR A) -> VARBINARY(16) B

If A has no colons it is prepended with '::fff:' If A is NULL, too long, or if inet_pton returns an error, the result is NULL.

Examples

=> SELECT V6_ATON('2001:DB8::8:000:200C:417A');
  v6_aton
  -----------
  \001\015\270\000\000\000\000\010\010\000 \014Az
  (1 row)

=> SELECT V6_ATON('1.2.3.4');
  v6_aton
  -----------
  \000\000\000\000\000\000\000\000\377\377\001\002\003\004
  (1 row)
SELECT TO_HEX(V6_ATON('2001:DB8::8:000:200C:417A'));
  to_hex
  --------------------------
  20010db800000000000000002c417a
(1 row)

=> SELECT V6_ATON('::1.2.3.4');

  v6_aton
-------------------------------
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\001\002\003\004
(1 row)

See Also

- V6_NTOA

V6_NTOA

Converts an IPv6 address represented as varbinary to a character string.

Behavior Type

Immutable

Syntax

V6_NTOA ( expression )

Parameters

| expression | (VARBINARY) is the binary string to convert. |

Notes

The following syntax converts an IPv6 address represented as VARBINARY B to a string A. V6_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function inet_ntop.

=> V6_NTOA(VARBINARY B) -> VARCHAR A

If B is NULL or longer than 16 bytes, the result is NULL.
Vertica automatically converts the form ':fff:1.2.3.4' to '1.2.3.4'.

Examples

```sql
=> SELECT V6_NTOA('0010152700000000000000100100 \014Az');

v6_ntoa
---------------------
2001:db8::8:800:200c:417a
(1 row)

=> SELECT V6_NTOA(V6_ATON('1.2.3.4'));

v6_ntoa
--------
1.2.3.4
(1 row)

=> SELECT V6_NTOA(V6_ATON('::1.2.3.4'));

v6_ntoa
--------
::1.2.3.4
(1 row)
```

See Also

- V6_ATON

V6_SUBNETA

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

`V6_SUBNETA ( expression1, expression2 )`
Parameters

| expression1 | (VARBINARY or VARCHAR) is the string to calculate. |
| expression2 | (INTEGER) is the size of the subnet. |

Notes

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

```sql
=> V6_SUBNETA(BINARY B, INT8 N) -> VARCHAR C
```

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

```sql
=> V6_SUBNETA(VARCHAR A, INT8 N) -> V6_SUBNETA(V6_ATON(A), N) -> VARCHAR C
```

Examples

```sql
=> SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
V6_subneta
 -----------------
2001:db0::/28
(1 row)
```

See Also

- V6_SUBNETN

V6_SUBNETN

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.
Behavior Type

Immutable

Syntax

V6_SUBNETN ( expression1, expression2 )

Parameters

| expression1 | (VARBINARY or VARCHAR) is the string to calculate. |
| Notes: | |
| | • V6_SUBNETN(<VARBINARY>, <INTEGER>) returns VARBINARY. |
| OR | |
| | • V6_SUBNETN(<VARCHAR>, <INTEGER>) returns VARBINARY, after using V6_ATON to convert the <VARCHAR> string to <VARBINARY>. |

| expression2 | (INTEGER) is the size of the subnet. |

Notes

The following syntax masks a BINARY IPv6 address B so that the N left-most bits of S form a subnet address, while the remaining right-most bits are cleared.

V6_SUBNETN right-pads B to 16 bytes with zeros, if necessary and masks B, preserving its N-bit subnet prefix.

=> V6_SUBNETN(VARBINARY B, INT8 N) -> VARBINARY(16) S

If B is NULL or longer than 16 bytes, or if N is not between 0 and 128 inclusive, the result is NULL.

S = [B]/N in Classless Inter-Domain Routing notation (CIDR notation).

The following syntax masks an alphanumeric IPv6 address A so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared.
Example

This example returns VARBINARY, after using V6_ATON to convert the VARCHAR string to VARBINARY:

```
=> SELECT V6_SUBNETN(V6_ATON('2001:db8::8:800:200c:417a'), 28);

 v6_subnetn
--------------------------
 \x001\015\260\000\000\000\000\000\000\000\000\000\000
```

See Also

- V6_ATON
- V6_SUBNETA

V6_TYPE

Characterizes a binary or alphanumeric IPv6 address B as an integer type.

Behavior Type

Immutable

Syntax

```
V6_TYPE ( expression )
```

Parameters

| expression | (VARBINARY or VARCHAR) is the type to convert. |
Notes

V6_TYPE(VARBINARY B) returns INT8 T.

=> V6_TYPE(VARCHAR A) -> V6_TYPE(V6_ATON(A)) -> INT8 T

The IPv6 types are defined in the Network Working Group's IP Version 6 Addressing Architecture memo.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL = 0</td>
<td>Global unicast addresses</td>
</tr>
<tr>
<td>LINKLOCAL = 1</td>
<td>Link-Local unicast (and Private-Use) addresses</td>
</tr>
<tr>
<td>LOOPBACK = 2</td>
<td>Loopback</td>
</tr>
<tr>
<td>UNSPECIFIED = 3</td>
<td>Unspecified</td>
</tr>
<tr>
<td>MULTICAST = 4</td>
<td>Multicast</td>
</tr>
</tbody>
</table>

IPv4-mapped and IPv4-compatible IPv6 addresses are also interpreted, as specified in IPv4 Global Unicast Address Assignments.

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

Details

IPv4 (either kind):

<table>
<thead>
<tr>
<th>IPv4</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0/8</td>
<td>UNSPECIFIED</td>
</tr>
<tr>
<td>127.0.0.0/8</td>
<td>LOOPBACK</td>
</tr>
<tr>
<td>169.254.0.0/16</td>
<td>LINKLOCAL</td>
</tr>
<tr>
<td>172.16.0.0/12</td>
<td>LINKLOCAL</td>
</tr>
<tr>
<td>192.168.0.0/16</td>
<td>LINKLOCAL</td>
</tr>
<tr>
<td>224.0.0.0/4</td>
<td>MULTICAST</td>
</tr>
<tr>
<td>others</td>
<td>GLOBAL</td>
</tr>
</tbody>
</table>

IPv6:

<table>
<thead>
<tr>
<th>IPv6</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>::0/128</td>
<td>UNSPECIFIED</td>
</tr>
<tr>
<td>fe80::/10</td>
<td>LINKLOCAL</td>
</tr>
<tr>
<td>ff00::/8</td>
<td>MULTICAST</td>
</tr>
<tr>
<td>others</td>
<td>GLOBAL</td>
</tr>
</tbody>
</table>
Examples

```sql
=> SELECT V6_TYPE(V6_ATON('192.168.2.10'));
   v6_type
  --------
      1
   (1 row)

=> SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
   v6_type
  --------
       0
   (1 row)
```

See Also

- INET_ATON
- IP Version 6 Addressing Architecture
- IPv4 Global Unicast Address Assignments

Machine Learning Functions

The machine learning functions contain algorithms for machine learning and data preparation. Additionally, these functions provide evaluation metrics for models. You can use these evaluation metrics to determine the accuracy of your models.

Vertica machine learning functions do not support temp tables.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**APPLY_KMEANS**

Assigns each row of an input table or view to a cluster center from an already-existing k-means model.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.
Syntax

APPLY_KMEANS ( col1, col2, ..., coln

      USING PARAMETERS model_name='model_name'
      [,match_by_pos = 'method'] )

Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

Parameters

| model_name='model_name' | The name of the k-means model you created previously. Model names are case-insensitive. |
| match_by_pos='method' | (Optional) Valid Values: |
|                       | • false (default): Input columns will be matched to features in the model based on their names. |
|                       | • true: Input columns will be matched to features in the model based on their position in the list of indicated input columns. |

Privileges

To use APPLY_KMEANS, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

The following example shows how you can use the APPLY_KMEANS function on an input table. Note that you can mix column names and constants:

=> SELECT id, APPLY_KMEANS(Sepal_Length, 2.2, 1.3,
       Petal_Width USING PARAMETERS
       model_name='myKmeansModel', match_by_pos='true') FROM iris2;
The following example shows how you can use the APPLY_KMEANS function on an input table, using the match_by_pos parameter. Note that providing constants instead of column names works with this parameter:

```sql
=> SELECT id, APPLY_KMEANS(0,0,0,0 USING PARAMETERS
               model_name='myKmeansModel', match_by_pos='true')
       FROM iris ORDER BY id;

<table>
<thead>
<tr>
<th>id</th>
<th>APPLY_KMEANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>
```

(150 rows)
See Also

- Clustering Data Using k-means
- KMEANS

APPLY_NORMALIZE

Applies the normalization parameters saved in a model to a set of specified columns in the input table or view. If any column specified in the function is not contained in the model, the data will pass through unchanged to APPLY_NORMALIZE.

Syntax

```
APPLY_NORMALIZE ( input_columns
   USING PARAMETERS model_name='model_name');
```

Arguments

| input_columns | The columns to use from the input table or view. Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of column names, the function selects all the columns in the input table or view, but normalizes all columns present in the model. |

Parameters

| model_name='model_name' | The name of the model. Model names are case-insensitive. Value must be VARCHAR. |

Privileges

To use APPLY_NORMALIZE, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.
See [GRANT (Schema)](Vertica documentation) and [GRANT (Table)](Vertica documentation).

### Examples

This example shows how you can use the `APPLY_NORMALIZE` function on the `hp` and `cyl` columns in the `mtcars` table, where `hp` is in the normalization model and `cyl` is not in the normalization model.

```sql
=> SELECT APPLY_NORMALIZE (hp, cyl USING PARAMETERS model_name = 'mtcars_normfit') FROM mtcars;
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hp</td>
<td>cyl</td>
</tr>
<tr>
<td>0.62897527217865</td>
<td>8</td>
</tr>
<tr>
<td>0.681978821754456</td>
<td>8</td>
</tr>
<tr>
<td>0.434628963470459</td>
<td>8</td>
</tr>
<tr>
<td>0.575971722602844</td>
<td>8</td>
</tr>
<tr>
<td>0.20494694781494</td>
<td>6</td>
</tr>
<tr>
<td>0.20494694781494</td>
<td>6</td>
</tr>
<tr>
<td>0.681978821754456</td>
<td>8</td>
</tr>
<tr>
<td>0.434628963470459</td>
<td>6</td>
</tr>
<tr>
<td>0.049469664413929</td>
<td>4</td>
</tr>
<tr>
<td>0.049469664413929</td>
<td>4</td>
</tr>
<tr>
<td>0.749116599559784</td>
<td>8</td>
</tr>
<tr>
<td>0.0353356897830963</td>
<td>4</td>
</tr>
<tr>
<td>0.452296823263168</td>
<td>8</td>
</tr>
<tr>
<td>0.159010604023933</td>
<td>4</td>
</tr>
<tr>
<td>0.346289753913879</td>
<td>8</td>
</tr>
<tr>
<td>0.54063606262207</td>
<td>8</td>
</tr>
<tr>
<td>0.144876331090927</td>
<td>4</td>
</tr>
<tr>
<td>0.346289753913879</td>
<td>8</td>
</tr>
<tr>
<td>0.20494694781494</td>
<td>6</td>
</tr>
<tr>
<td>0.215547695755959</td>
<td>4</td>
</tr>
<tr>
<td>0.1519436010685</td>
<td>4</td>
</tr>
<tr>
<td>0.25088340040216</td>
<td>6</td>
</tr>
<tr>
<td>0.25088340040216</td>
<td>6</td>
</tr>
<tr>
<td>0.452296823263168</td>
<td>8</td>
</tr>
<tr>
<td>0.452296823263168</td>
<td>8</td>
</tr>
<tr>
<td>0.434628963470459</td>
<td>8</td>
</tr>
<tr>
<td>0.137809187173843</td>
<td>4</td>
</tr>
<tr>
<td>0.045936394482851</td>
<td>4</td>
</tr>
<tr>
<td>0.187279149889946</td>
<td>6</td>
</tr>
</tbody>
</table>

(32 rows)

See Also

- [NORMALIZE](Vertica documentation)
- [NORMALIZE_FIT](Vertica documentation)
Normalizing Data

**REVERSE_NORMALIZE**

**APPLY_ONE_HOT_ENCODER**

A user-defined transform function (UDTF) that loads the model and writes out a table that contains the encoded columns.

**Syntax**

```
APPLY_ONE_HOT_ENCODER( 'input_columns'
  USING PARAMETERS
  model_name,
  [drop_first='false | true'],
  [ignore_null='false | true'],
  [separator='method'])
```

**Arguments**

<table>
<thead>
<tr>
<th>input_columns</th>
<th>A comma-separated list of the columns for one hot encoding. At least one column is required. Supports the use of wildcard (<em>) characters in place of column names. If you use a wildcard character (</em>) in place of column names, the function selects all the columns in the input table or view, with the additional indicator columns for each column present in the model.</th>
</tr>
</thead>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>model_name</th>
<th>The name of the model where the categories and their corresponding levels are stored. Model names are case-insensitive.</th>
</tr>
</thead>
</table>

| drop_first='false | true' | (Optional) If False, every level of the categorical variable has a corresponding column in the output view. If True, the first level of the categorical variable is treated as the reference level. |

**Default Value:** True

| ignore_null='false | true' | (Optional) If False, NULL values in the input_columns are treated as a categorical level. If True, NULL values set all |

...
corresponding one-hot binary columns to NULL.

Default Value: True.

separator = 'method'
(Optional) Denotes the separator character between the input variable name and the indicator variable level in the output table. Use NULL to avoid using any separator.

Default Value: '_'

Note: When a level not stored in the model appears in an input row, the columns corresponding to that categorical level in the output row are returned as NULL values.

Privileges

To use APPLY_ONE_HOT_ENCODER, you must either be a superuser or have CREATE privileges for the schema and SELECT privileges for the table.

See GRANT (Schema) and GRANT (Table).

Examples

The following example shows how you can use the APPLY_ONE_HOT_ENCODER function:

```
=> SELECT APPLY_ONE_HOT_ENCODER(cyl USING PARAMETERS model_name='one_hot_encoder_model',
   drop_first='true', ignore_null='false') FROM mtcars;
  cyl | cyl_1 | cyl_2
-----------------------
   8 |   0 |   1
   4 |   0 |   0
   4 |   0 |   0
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   8 |   0 |   1
   6 |   1 |   0
   4 |   0 |   0
   4 |   0 |   0
   6 |   1 |   0
   6 |   1 |   0
   8 |   0 |   1
   8 |   0 |   1
   4 |   0 |   0
   4 |   0 |   0
   6 |   1 |   0
   6 |   1 |   0
```

Vertica Analytic Database (9.0.x)
See Also

- Encoding Categorical Columns
- ONE_HOT_ENCODER_FIT

**BALANCE**

Returns a view with an equal distribution of the input data based on the dependent variable.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```
BALANCE ( 'output_view', 'input_relation', 'response_column', 'balance_method'
           [USING PARAMETERS [sampling_ratio=value] ])
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'output_view'</td>
<td>The name of the View where Vertica saves the balanced data from the chosen input_relation.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The output_view that results from this function employs a random function. Its content can differ each time it is used in a query. To make the operations on the output_view predictable,</td>
</tr>
</tbody>
</table>
store it in a regular table.

### 'input_relation'
The table or view that contains the data the function uses to create a more balanced data set. If the `input_relation` is defined in Hive, you must sync the hcatalog schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function.

### 'response_column'
The dependent variable: a VARCHAR or INTEGER column in the `input_relation`.

### 'balance_method'
The imbalanced processing method to use. Selects data from the minority and majority classes.

#### Valid values
- hybrid_sampling: Performs oversampling and undersampling on different classes so each class is equally represented.
- over_sampling: Oversamples on all classes, with the exception of the most majority class, towards the most majority class's cardinality.
- under_sampling: Undersamples on all classes, with the exception of the most minority class, towards the most minority class's cardinality. An alias of weighted_sampling.
- weighted_sampling: An alias of under_sampling.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sampling_ratio=value</code></td>
<td>The desired ratio between the majority class and the minority class. This value has no effect when used with balance method hybrid_sampling.</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: 1.0</td>
</tr>
</tbody>
</table>

### Privileges

To use BALANCE, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT (Schema)] and [GRANT (Table)].
Examples

The following example shows how you can use the BALANCE function:

```sql
--> CREATE TABLE backyard_bugs (id identity, bug_type int, finder varchar(20));
CREATE TABLE

--> COPY backyard_bugs FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

   >> 1|Ants
   >> 1|Beetles
   >> 3|Ladybugs
   >> 3|Ants
   >> 3|Beetles
   >> 3|Caterpillars
   >> 2|Ladybugs
   >> 3|Ants
   >> 3|Beetles
   >> 1|Ladybugs
   >> 3|Ladybugs
   >> 

--> SELECT bug_type, COUNT(bug_type) FROM backyard_bugs GROUP BY bug_type;

bug_type | COUNT
---------|------
   2 |  1
   1 |  3
   3 |  7
(3 rows)

--> SELECT BALANCE('backyard_bugs_balanced', 'backyard_bugs', 'bug_type', 'under_sampling');

BALANCE
Finished in 1 iteration

(1 row)

--> SELECT bug_type, COUNT(bug_type) FROM backyard_bugs_balanced GROUP BY bug_type;

---------|------
   2 |  1
   1 |  2
   3 |  1
(3 rows)
```

See Also

- Balancing Imbalanced Data
CONFUSION_MATRIX

Using an input table, returns a confusion matrix based on observed and predicted values. CONFUSION_MATRIX produces a table with the following dimensions:

- Rows: Number of classes
- Columns: Number of classes + 2

Syntax

CONFUSION_MATRIX ( targets, predictions
[ USING PARAMETERS [num_classes=C] ]
OVER() )

Arguments

<table>
<thead>
<tr>
<th>targets</th>
<th>An integer input column that contains the true values of the response variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>predictions</td>
<td>An integer input column that contains the predicted class labels</td>
</tr>
</tbody>
</table>

Both input columns identify classes as zero-based consecutive integers between 0 and (num-classes-1) inclusive, where num-classes is the number of classes. For example, given the following input column values— {0, 1, 2, 3, 4}—Vertica assumes five classes.

Note: If input column values are not consecutive, Vertica interpolates the missing values. Thus, given the following input values— {0, 1, 3, 5, 6}—Vertica assumes seven classes.

Parameters

| num_classes=C | An integer ≥ 1, specifies the number of classes to pass to the function. By default set to 2, you must set this parameter correctly if the number of classes is any other value. |
Examples

This example shows how you can execute the CONFUSION_MATRIX function on an input table named mtcars. The response variables appear in the column obs, while the prediction variables appear in the column pred. Because this problem regards classification, both the response variable values and the prediction variable values are either 0 or 1, indicating binary classification.

In the table returned, all 13 cars with a value of 1 in the am column were correctly predicted by PREDICT_LOGISTIC_REG as having a value of 1. Out of the 20 cars that had a value of 0 in the am column, 19 were correctly predicted to have the value 0. One car was incorrectly classified as having the value 1.

```sql
=> SELECT CONFUSION_MATRIX(obs::int, pred::int USING PARAMETERS num_classes=2) OVER(
    FROM (SELECT am AS obs, PREDICT_LOGISTIC_REG(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb USING PARAMETERS model_name='myLogisticRegModel')AS pred
    FROM mtcars) AS prediction_output;

<table>
<thead>
<tr>
<th>class</th>
<th>0</th>
<th>1</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>19</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
| 1     | 1  | 13 | Of 32 rows, 32 were used and 0 were ignored
```

CROSS_VALIDATE

Performs cross validation on a learning algorithm using an input relation, and performs grid search for hyper parameters. The output is an average performance indicator of the selected algorithm. This function only supports SVM classification, naive bayes, and logistic regression.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```sql
CROSS_VALIDATE ( 'algorithm', 'input_relation', 'response_column', 'predictor_columns'
   [ USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
   [cv_model_name='string',]
   [cv_metrics='value',]
   [cv_fold_count=value,]
   [cv_hyperparams='json_string',]
   [cv_prediction_cutoff=cutoff_value,]])
```
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithm</code></td>
<td>The name of the training function of the algorithm.</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Values:</strong> svm_classifier, naive_bayes, or logistic_reg</td>
</tr>
<tr>
<td><code>input_relation</code></td>
<td>The table or view that contains the data used for training and testing. If the input_relation is defined in Hive, you must sync the hcatalog schema using the <code>SYNC_WITH_HCATALOG_SCHEMA</code> function, and then run the machine learning function.</td>
</tr>
<tr>
<td><code>response_column</code></td>
<td>The name of the column in the input_relation that contains the response.</td>
</tr>
<tr>
<td><code>predictor_columns</code></td>
<td>A comma-separated list of the columns in the input_relation that are passed to the algorithm as predictors. Supports the use of wildcard (<em>) characters in place of column names. If you use a wildcard character (</em>) in place of a column name, all the columns in input_relation are selected.</td>
</tr>
</tbody>
</table>
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude_columns='col1, col2, ... coln'</td>
<td>(Optional) The columns from input_relation that you want to exclude from the predictor_columns argument.</td>
</tr>
<tr>
<td></td>
<td>Default Value: Empty</td>
</tr>
<tr>
<td>cv_model_name= 'string'</td>
<td>(Optional) The name that is used to retrieve the result of the cross validation process. If you do not supply this parameter, the results are displayed, but they are not saved. If the parameter is supplied, you can retrieve the results again by using the summary functions GET_MODEL_SUMMARY or GET_MODEL_ATTRIBUTE.</td>
</tr>
<tr>
<td>cv_metrics= 'value'</td>
<td>(Optional) A comma-separated list of metrics used to assess the algorithm.</td>
</tr>
<tr>
<td></td>
<td>Valid Values: Accuracy and error_rate</td>
</tr>
<tr>
<td></td>
<td>Default Value: Accuracy</td>
</tr>
<tr>
<td>cv_fold_count=value</td>
<td>(Optional) The number of folds to split the data into.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 5</td>
</tr>
<tr>
<td>cv_hyperparams='json_string'</td>
<td>(Optional) A JSON string that describes the combination of parameters for use in grid search of hyper parameters. The JSON string contain pairs of the hyper parameter name. The value of each hyper parameter can be specified as an array or sequence such as:</td>
</tr>
<tr>
<td></td>
<td>{&quot;param1&quot;:[value1,value2,...], &quot;param2&quot;:{&quot;first&quot;:first_value, &quot;step&quot;:step_size, &quot;count&quot;:number_of_values} }</td>
</tr>
<tr>
<td></td>
<td>Note that hyper parameter names and string values should be quoted using the JSON standard. These parameters are passed to the training function.</td>
</tr>
<tr>
<td>cv_prediction_cutoff=cutoff_value</td>
<td>(Optional) The cutoff threshold that is passed to the prediction stage of logistic regression.</td>
</tr>
<tr>
<td></td>
<td>Valid Values: [0,1]</td>
</tr>
</tbody>
</table>
Vertica Documentation
SQL Reference Manual

Default Value: 0.5

Privileges
To use CROSS_VALIDATE, you must either be a superuser or have CREATE and USAGE privileges
for the schema where the algorithm generates models. If cv_model_name is provided, the
cross validation results are also saved as a model in that same schema. There are no privileges
needed on the function itself.
See GRANT (Schema) and GRANT (Table).

Examples
This example shows how you can use the CROSS_VALIDATE function.
=> SELECT CROSS_VALIDATE('svm_classifier', 'mtcars', 'am', 'mpg' USING PARAMETERS
cv_fold_count= 6, cv_hyperparams='{"C":[1,5]}', cv_model_name='cv_svm',
cv_metrics='accuracy,error_rate');
CROSS_VALIDATE
---------------------------Finished
===========
run_average
===========
C |accuracy
|error_rate
---+--------------+---------1 | 0.75556
| 0.24444
5 | 0.78333
| 0.21667
(1 row)

This example shows how you can use the GET_MODEL_ATTRIBUTE on the model after you
apply cross validation:
=> SELECT GET_MODEL_ATTRIBUTE(USING PARAMETERS attr_name='details',
model_name='cv_svm');
attr_name
|
attr_fields
| #_of_rows
------------+---------------------------------------------------+----------run_details | C, fold_id, iteration_count, accuracy, error_rate |
12
run_average | C, accuracy, error_rate
|
2
fold_info
| fold_id, row_count
|
6
counters
| counter_name, counter_value
|
3
call_string | call_string
|
1
(5 rows)

Vertica Analytic Database (9.0.x)

Page 2660 of 6180


See Also

- Cross Validation Attributes

**DETECT_OUTLIERS**

Returns the outliers in a data set based on the outlier threshold. The output is a table containing the outliers.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Behavior Type**

Immutable

**Syntax**

```
DETECT_OUTLIERS ( 'output_table', 'input_relation', 'input_columns', 'outlier_method'
    USING PARAMETERS outlier_threshold=value
    [, exclude_columns='col1, col2, ... coln',]
    [partition_columns='col1, col2, ... coln'])
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'output_table'</td>
<td>The name of the table where Vertica saves the outliers from the chosen input_columns.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains outlier data. If the input relation is defined in Hive, you must sync the hcatalog schema using the <code>SYNC_WITH_HCATALOG_SCHEMA</code> function, and then run the machine learning function.</td>
</tr>
</tbody>
</table>
The columns of input_relation to be used for determining outliers. The input_columns argument supports the use of a wildcard (*) character in place of column names.

The outlier method to use:

Valid Values:
- robust_zscore

Parameters

- **outlier_threshold=value** (Optional) The value beyond which a data point becomes an outlier.
  
  Default value: 3.0

- **exclude_columns='col1, col2, ... col'** (Optional) The columns from input_relation which you want to exclude from the input_columns argument.

- **partition_columns='col1, col2, ... coln'** (Optional) A comma-separated list of column names from the input_relation which defines the partitions. The function will detect outliers among each partition separately. The default value is empty.

Privileges

To use DETECT_OUTLIERS, you must either be a superuser or have CREATE privileges for the schema and SELECT privileges for the table.

See **GRANT (Schema)** and **GRANT (Table)**.

Examples

The following example shows how you can use the DETECT_OUTLIERS function:
CREATE TABLE baseball_roster (id identity, last_name varchar(30), hr int, avg float);

COPY baseball_roster FROM STDIN;

Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.

> Polo|7|.233
> Gloss|45|.170
> Gus|12|.345
> Gee|1|.125
> Laus|3|.095
> Hilltop|16|.222
> Wicker|78|.333
> Scooter|0|.121
> Hank|999999|.8888
> Popup|35|.378
> 

SELECT *
FROM baseball_roster;

<table>
<thead>
<tr>
<th>id</th>
<th>last_name</th>
<th>hr</th>
<th>avg</th>
</tr>
</thead>
</table>
| 3  | Gus       | 12 | 0.345
| 4  | Gee       | 1  | 0.125
| 6  | Hilltop   | 16 | 0.222
| 10 | Popup     | 35 | 0.378
| 1  | Polo      | 7  | 0.233
| 7  | Wicker    | 78 | 0.333
| 9  | Hank      | 999999 | 0.8888
| 2  | Gloss     | 45 | 0.17 
| 5  | Laus      | 3  | 0.095
| 8  | Scooter   | 0  | 0.121

(10 rows)

SELECT DETECT_OUTLIERS('baseball_outliers', 'baseball_roster', 'id, hr, avg', 'robust_zscore' USING PARAMETERS outlier_threshold=3.0);

DETECT_OUTLIERS

Detected 2 outliers

(1 row)

SELECT *
FROM baseball_outliers;

<table>
<thead>
<tr>
<th>id</th>
<th>last_name</th>
<th>hr</th>
<th>avg</th>
</tr>
</thead>
</table>
| 7  | Wicker    | 78 | 0.333
| 9  | Hank      | 999999 | 0.8888

(2 rows)

ERROR_RATE

Using an input table, returns a table which calculates the rate of incorrect classifications. ERROR_RATE produces a table with the following dimensions:
- Rows: Number of classes
- Columns: Number of classes + 2

**Syntax**

```sql
ERROR_RATE (targets, predictions
    [ USING PARAMETERS [num_classes=C] ])
OVER()
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>targets</code></td>
<td>An integer input column that contains the true values of the response variable.</td>
</tr>
<tr>
<td><code>predictions</code></td>
<td>An integer input column that contains the predicted class labels</td>
</tr>
</tbody>
</table>

Both input columns identify classes as zero-based consecutive integers between 0 and (num-classes-1) inclusive, where `num-classes` is the number of classes. For example, given the following input column values— {0, 1, 2, 3, 4}—Vertica assumes five classes.

**Note:** If input column values are not consecutive, Vertica interpolates the missing values. Thus, given the following input values— {0, 1, 3, 5, 6}—Vertica assumes seven classes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_classes=C</code></td>
<td>An integer ≥ 1, specifies the number of classes to pass to the function. By default set to 2, you must set this parameter correctly if the number of classes is any other value.</td>
</tr>
</tbody>
</table>

**Privileges**

To use `ERROR_RATE`, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See [GRANT (Schema)](https://www.vertica.com/documentation) and [GRANT (Table)](https://www.vertica.com/documentation).
Examples

This example shows how you can execute the ERROR_RATE function on an input table named mtcars. The response variables appear in the column obs, while the prediction variables appear in the column pred. Because this example is a classification problem, all of the response variable values and the prediction variable values are either 0 or 1, indicating binary classification.

In the table returned by the function, the first column displays the class id column. The second column displays the corresponding error rate for the class id. The third column indicates how many rows were successfully used by the function and whether any rows were ignored.

```
=> SELECT ERROR_RATE(obs::int, pred::int USING PARAMETERS num_classes=2) OVER()
    FROM (SELECT am AS obs, PREDICT_LOGISTIC_REG (mpg, cyl, disp, drat, wt, qsec, vs, gear, carb
           USING PARAMETERS model_name='myLogisticRegModel', type='response') AS pred
    FROM mtcars) AS prediction_output;
```

<table>
<thead>
<tr>
<th>class</th>
<th>error_rate</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0769230797886848</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.03125</td>
<td>Of 32 rows, 32 were used and 0 were ignored</td>
</tr>
</tbody>
</table>

(3 rows)

EXPORT_MODELS

Exports model(s) to another Vertica cluster.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

EXPORT_MODELS ( 'destination', 'scope' )

Arguments

| 'destination' | Specifies where to store the exported model. Must be the absolute path of a SQL output file. |
| 'scope' | Specifies the machine learning model(s) to export in the format [schema].model. If you specify a file that does not exist, the |
Privileges

To use EXPORT_MODELS, you must be a superuser.

Examples

The following example shows how you can use the EXPORT_MODELS function to export the model named mykmeansmodel:

```
=> SELECT EXPORT_MODELS ('/home/dbadmin', 'myschema.mykmeansmodel')
EXPORT_MODELS
--------------
Success
(1 row)
```

The following example shows how you can use the EXPORT_MODELS function to export all models under the myschema destination:

```
=> SELECT EXPORT_MODELS ('/home/dbadmin', 'myschema.*')
EXPORT_MODELS
--------------
Success
(1 row)
```

GET_MODEL_ATTRIBUTE

Extracts either a specific attribute from a model or all attributes from a model. Use this function to view a list of attributes and row counts or view detailed information about a single attribute. The output of GET_MODEL_ATTRIBUTE is a table format where users can select particular columns or rows.

Syntax

```
GET_MODEL_ATTRIBUTE (USING PARAMETERS model_name='model name',
[  attr_name='value']
)```

Parameters

<table>
<thead>
<tr>
<th>model_name='model name'</th>
<th>The name of the model. Model names are case-insensitive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr_name</td>
<td>(Optional) The name of the model attribute. If the value is not specified, the function shows all available attributes. Attribute names are case-sensitive.</td>
</tr>
</tbody>
</table>

Privileges

To use GET_MODEL_ATTRIBUTE, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can view a summary of all available attributes for a model.

```sql
=> SELECT GET_MODELATTRIBUTE (USING PARAMETERS model_name='myLinearRegModel');
attr_name | attr_fields | #_of_rows
----------|-------------|---------
data      | coeffNames, coeff, stdErr, zValue, pValue | 2
regularization | type, lambda | 1
```
This example shows how you can extract the data attribute from the myLinearRegModel model.

```
=> SELECT GET_MODEL_ATTRIBUTE (USING PARAMETERS model_name='myLinearRegModel', attr_name='data');</code><pre>
coeffNames | coeff | stdErr | zValue | pValue

Intercept | -1.87401598641074 | 0.160143331525544 | -11.7021169008952 | 7.3592939615234e-26
waiting | 0.0756279479518627 | 0.00221854185633525 | 34.0890336307608 | 8.13028381124448e-100
(2 rows)
```

**GET_MODEL_SUMMARY**

Returns the summary information of a model.

**Syntax**

```
GET_MODEL_SUMMARY (USING PARAMETERS model_name= 'model_name')
```

**Parameters**

- **model_name= 'model_name'**
  
  The name of the model you want to summarize. Model names are case-insensitive.

**Privileges**

To use GET_MODEL_SUMMARY, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

**Examples**

This example shows how you can view the summary of a linear regression model.

```
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='myLinearRegModel');
```
details
predictor|coefficient|std_err  |t_value |p_value
----------|-----------|---------|--------|--------
Intercept| -2.06795  | 0.21063 | 9.81782| 0.00000
waiting   | 0.07876   | 0.00292 | 26.96925| 0.00000

regularization

lambda

none|1.00000

call_string

linear_reg('public.linear_reg_faithful', 'faithful_training', 'eruptions', 'waiting'
USING PARAMETERS optimizer='bfgs', epsilon=1e-06, max_iterations=100,
regularization='none', lambda=1)

Additional Info

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>iteration_count</td>
<td>3</td>
</tr>
<tr>
<td>rejected_row_count</td>
<td>0</td>
</tr>
<tr>
<td>accepted_row_count</td>
<td>162</td>
</tr>
</tbody>
</table>

(1 row)

IMPORT_MODELS

Import model(s) from another Vertica cluster. The import model name must match the name of the exported model. If you have a naming conflict, you must import the model using a different schema, and then rename the model.

Caution: Changing the exported model files causes the import functionality to fail.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

IMPORT_MODELS ( 'source'
    USING PARAMETERS new_schema='schema_name')
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'source'</td>
<td>Specifies the location to import the model from. Must be the absolute path of a SQL output file. If the source ends with '*', all models under this source are imported.</td>
</tr>
<tr>
<td>'schema_name'</td>
<td>Specifies the schema where the machine learning model(s) will be imported. This schema must be created before you import the models.</td>
</tr>
</tbody>
</table>

Privileges

To use IMPORT_MODELS, you must be a superuser.

Examples

The following example shows how you can use the IMPORT_MODELS function to import the model named mykmeansmodel:

```sql
=> SELECT IMPORT_MODELS ('/home/dbadmin/myschema/mykmeansmodel' USING PARAMETERS new_schema='newschema')
IMPORT_MODELS
--------------
Success
(1 row)
```

The following example shows how you can use the IMPORT_MODELS function and import all models under the myschema destination:

```sql
=> SELECT IMPORT_MODELS ('/home/dbadmin/myschema/*' USING PARAMETERS new_schema='newschema')
IMPORT_MODELS
--------------
Success
(1 row)
```

**IMPUTE**

Imputes missing values in a data set with either the mean or the mode, based on observed values for a variable. This function supports both numeric and categorical data types.
Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

IMPUTE('output_view', 'input_relation', 'input_columns', 'method'
    [ USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
    [partition_columns='col1, col2, ... coln'] ])}

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'output_view'</td>
<td>The name of the View where the missing-value imputed rows and rows without missing values are stored. The value must be VARCHAR.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains the data for missing-value imputation. The value must be VARCHAR. If the input_relation is defined in Hive, you must sync the hcatalog schema using the SYNC_WITH_HCATALOG_SCHEMA function, and then run the machine learning function.</td>
</tr>
<tr>
<td>'input_columns'</td>
<td>A comma-separated list of the columns in input_relation containing the values used in missing value imputation. The value must be VARCHAR. Supports the use of the * character.</td>
</tr>
<tr>
<td>'method'</td>
<td>The missing-value</td>
</tr>
</tbody>
</table>
imputation method to use.

Valid Values:
- mean: for numeric missing-value imputation
- mode: for categorical missing-value imputation

Numeric and categorical input values are supported.

### Parameters

| exclude_columns='col1, col2, ... coln' | (Optional) The columns from input_relation that you want to exclude from the input_columns argument. Default Value: Empty |
| partition_columns='col1, col2, ... coln' | (Optional) A comma-separated list of column names from input_relation for the partition clause. Default Value: Empty |

### Privileges

To use IMPUTE, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See [GRANT (Schema)](https://www.vertica.com/docs/9.0.x/HTML/R007014_000826.html) and [GRANT (Table)](https://www.vertica.com/docs/9.0.x/HTML/R007014_000826.html).

### Examples

These examples show how you can use the IMPUTE function on the `small_input_impute` table.

Execute the IMPUTE function, specifying the mean method:
Execute the IMPUTE function, specifying the mode method:

```
=> SELECT impute('output_view3','small_input_impute', 'pid, x5,x6','mode' USING PARAMETERS exclude_columns='pid');
impute
------------------------
Finished in 1 iteration
 (1 row)
```

See Also

Imputing Missing Values

KMEANS

Executes the k-means algorithm on an input table or view. The result is a model with a list of cluster centers.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```
KMEANS ( 'model_name', 'input_relation', 'input_columns', num_clusters
        [ USING PARAMETERS [exclude_columns=['col1, col2, ... coln'],
        [max_iterations=value,]
        [epsilon=value,]
        [init_method='method',]
        [initial_centers_table='table_name',]
        [output_view='output_view',]
        [key_columns='key_columns'] ])
```

Arguments

| 'model_name' | Identifies the model, where |
**model_name** conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.

**'input_relation'**

The table or view that contains the input data for k-means. If the input_relation is defined in Hive, you must sync the hcatalog schema using the **SYNC_WITH_HCATALOG_SCHEMA** function, and then run the machine learning function.

**'input_columns'**

The columns of input_relation to be used for clustering. The input_columns argument supports the use of wildcard (*) characters in place of column names.

**'num_clusters'**

The number of clusters you want to create. This argument represents the $k$ in k-means. Must be an INTEGER and less than or equal to 10,000.

### Parameters

<p>| exclude_columns='col1, col2, ... coln' | (Optional) The columns from input_relation that you want to exclude from clustering. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max_iterations=value</code></td>
<td>(Optional) The maximum number of iterations the algorithm performs. If you set this value to a number lower than the number of iterations needed for convergence, the algorithm may not converge. Default Value: 10</td>
</tr>
<tr>
<td><code>epsilon=value</code></td>
<td>(Optional) Determines whether the algorithm has converged. The algorithm is considered converged after no center has moved more than a distance of 'epsilon' from the previous iteration. Default Value: 1e-4</td>
</tr>
</tbody>
</table>
| `init_method='method'`     | (Optional) The method used to find the initial cluster centers. You cannot use this parameter if the `initial_centers` parameter has a value. Providing a value for both `init_method` and `initial_centers_table` causes Vertica to return an error. Valid Values:  
  - random  
  - kmeanspp — kmeans++ algorithm. This value can be memory intensive for high k, so if you receive an error that not enough memory is available, either decrease the value of k or use the random method. Default Value: kmeanspp |
| `initial_centers_table='table_name'` | (Optional) The table with the initial cluster centers to use. Supply this value if you know the initial centers you want to use and do not want Vertica to find the initial cluster centers for you. You cannot use this parameter if the `init_method` parameter has a value. Providing a value for both `init_method` and `initial_centers_table` causes Vertica to return an error. |
| `output_view='output_view'` | (Optional) The name of the View where you save the assignments of each point to its cluster.                                                                                                               |
key_columns='key_columns'

(Optional) A comma-separated list of column names from the input_relation which you use to identify each row of the output in the output_view. Columns specified in key_columns should be also part of predictor_columns. To exclude any columns (including the columns specified in key_columns) from being used for prediction, they need to be listed in exclude_columns.

Privileges

To use KMEANS, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

The following example shows how you can use the KMEANS function and view the results of the model in the output_view.

```sql
=> SELECT KMEANS('myKmeansModel', 'iris1', '*', 5
  USING PARAMETERS max_iterations=20, output_view='myKmeansView', key_columns='id',
  exclude_columns='Species, id');

KMEANS

----------------------------------
Finished in 12 iterations

(1 row)
```

See Also

- Clustering Data Using k-means
- APPLY_KMEANS
- K-means Model Attributes
**LIFT_TABLE**

Returns a table that compares the predictive quality of a logistic regression model. This function is also known as a *lift chart*.

You cannot pass any inputs to the `OVER()` clause.

**Syntax**

```
LIFT_TABLE ( target, probabilities
            [ USING PARAMETERS [num_bins=nBins] ])
OVER()
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>target</code></td>
<td>The column in the input table containing the response variable. Must be an integer.</td>
</tr>
<tr>
<td><code>probabilities</code></td>
<td>The column in the input table where the observation is of class 1. Must be a float.</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_bins=nBins</code></td>
<td>(Optional) Groups rows together, based upon the probability column, for faster processing. You use this parameter to determine the number of different decision boundaries to consider. The parameter partitions the number line from 0 to 1 in <code>nBin</code> points that are equally spaced. It evaluates the table at each of the <code>nBin</code> points. Must be an integer.</td>
</tr>
<tr>
<td><strong>Default Value:</strong> 100</td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

This example demonstrates how you can execute the LIFT_TABLE function on an input table named `mtcars`.
The first column, `decision_boundary`, indicates the cut-off point for whether to classify a response as 0 or 1. For instance, for each row, if `prob` is greater than `decision_boundary`, the response is classified as 1. If `prob` is less than `decision_boundary`, the response is classified as 0.

The second column, `positive_prediction_ratio`, shows the percentage of samples in class 1 that the function classified correctly using the corresponding `decision_boundary` value.

For the third column, `lift`, the function divides the `positive_prediction_ratio` by the percentage of rows correctly classified as class 1.

### LINEAR_REG

Executes linear regression on an input table or view. The result is a linear regression model.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```sql
LINEAR_REG ( 'model_name', 'input_relation', 'response_column', 'predictor_columns' 
    [ USING PARAMETERS [exclude_columns='col1, col2, ... coln'],] 
    [optimizer='value',] 
    [epsilon=value,] 
    [max_iterations=value,] 
    [regularization= 'value',] 
    [lambda= value,] 
    [alpha = value]])
```
Arguments

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'model_name'</td>
<td>Identifies the model, where <em>model_name</em> conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains the training data for building the model. If the input_relation is defined in Hive, you must sync the hcatalog schema using the SYNC_WITH_HCATALOG_SCHEMA function, and then run the machine learning function.</td>
</tr>
<tr>
<td>'response_column'</td>
<td>The name of the column in the input_relation that represents the dependent variable, or outcome. For the model to be valid, all values in this column must be of type numeric.</td>
</tr>
<tr>
<td>'predictor_columns'</td>
<td>A comma-separated list of the columns in the input_relation that represent the independent variables for the model. For the model to be valid, all values in this column must be of type numeric. Supports the use of wildcard (<em>) characters in place of column names. If you use a wildcard character (</em>) in place of a column name, all the columns in input_relation are selected. In this case, the response_column must be explicitly excluded using the list assigned to exclude_columns.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude_columns='col1, col2, ... coln'</td>
<td>(Optional) The columns from input_relation that you want to exclude from the predictor_columns argument.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| optimizer='value'  | (Optional) The optimizer method used to train the model. If no optimizer is set and regularization is set to L1, the default optimizer switches to CGD. Valid Values:  
  - BFGS 
  - CGD (Coordinate Gradient Descent) - If this optimizer is not used in conjunction with regularization of L1 or ENet, Vertica throws an error that the regularization was selected but cannot be used. 
  - Newton (default value) |
| epsilon=value      | (Optional) Determines whether the algorithm has reached the specified accuracy result. Default Value: 1e-6                                                                                                      |
| max_iterations=value | (Optional) Determines the maximum number of iterations the algorithm performs before achieving the specified accuracy result. Default Value: 100                                                                |
| regularization='value' | (Optional) Determines the method of regularization. Default Value: None Valid Values:  
  - L1 
  - L2 
  - ENet 
  - None |
lambda=value

(Optional) The regularization parameter value. The value must be zero or positive.
Default Value: 1

alpha = value

(Optional) ENet mixture parameter that defines how much L1 versus L2 regularization to provide. This argument will send a warning if it is used without ENet regularization.
Valid Values: [0,1]
A value of 1 is equivalent to L1 and a value of 0 is equivalent to L2.

Privileges

To use LINEAR_REG, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the LINEAR_REG function.

=> SELECT LINEAR_REG('myLinearRegModel', 'faithful', 'eruptions', 'waiting'
    USING PARAMETERS optimizer='BFGS');

LINEAR_REG

--------------------------------------
Finished in 10 iterations

(1 row)

See Also

- Building a Linear Regression Model
- PREDICT_LINEAR_REG
Regression Model Attributes

LOGISTIC_REG

Executes logistic regression on an input table or view. The result is a logistic regression model.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

LOGISTIC_REG ('model_name', 'input_relation', 'response_column', 'predictor_columns'
[ USING PARAMETERS[exclude_columns='col1, col2, ... coln',]
[ optimizer='value',]  
[ epsilon=value, ]  
[ max_iterations=value ]  
[ regularization= 'value',]  
[ lambda= value,]  
[ alpha = value]])

Arguments

'\texttt{model\_name}'

Identifies the model, where \texttt{model\_name} conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.

'\texttt{input\_relation}'

The table or view that contains the training data for building the model. If
If the `input_relation` is defined in Hive, you must sync the `hcatalog` schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function.

| `'response_column'` | The name of the column in the `input_relation` that represents the dependent variable, or outcome.
For the model to be valid, all values in this column must be INTEGER types with a value of either 0 or 1. The function automatically skips all other values. |
| `'predictor_columns'` | A comma-separated list of the columns in the `input_relation` that represent the independent variables for the model.
For the model to be valid, all values in this column must be of type numeric.
Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, all the columns in `input_relation` are selected. In this case, the |
response_column must be explicitly excluded using the list assigned to exclude_columns.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude_columns='col1, col2, ... coln'</td>
<td>(Optional) The columns from input_relation that you want to exclude from the predictor_columns argument.</td>
</tr>
<tr>
<td>optimizer='value'</td>
<td>(Optional) The optimizer method to be used to train the model. Valid Values:</td>
</tr>
<tr>
<td></td>
<td>• BFGS</td>
</tr>
<tr>
<td></td>
<td>• CGD (Coordinate Gradient Descent) - If this optimizer is not used in conjunction with regularization of L1 or ENet, Vertica throws an error that the regularization was selected but cannot be used.</td>
</tr>
<tr>
<td></td>
<td>• Newton (default value)</td>
</tr>
<tr>
<td>epsilon=value</td>
<td>(Optional) Determines whether the algorithm has reached the specified accuracy result.</td>
</tr>
<tr>
<td>max_iterations=value</td>
<td>(Optional) Determines the maximum number of iterations the algorithm performs before achieving the specified accuracy result.</td>
</tr>
<tr>
<td>regularization='value'</td>
<td>(Optional) Determines the method of regularization.</td>
</tr>
</tbody>
</table>
Default Value: None

Valid Values:
- L1
- L2
- ENet
- None

\( \text{lambda}=value \)

(Optional) The regularization parameter value. The value must be zero or positive.
Default Value: 1

\( \text{alpha} = value \)

(Optional) ENet mixture parameter that defines how much L1 versus L2 regularization to provide. This argument will send a warning if it is used without ENet regularization.

Valid Values: [0,1]
A value of 1 is equivalent to L1 and a value of 0 is equivalent to L2.

Privileges

To use LOGISTIC_REG, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the LOGISTIC_REG function.

```sql
=> SELECT LOGISTIC_REG('myLogisticRegModel', 'mtcars', 'am',
                          'mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb')
```
USING PARAMETERS exclude_columns='hp', optimizer='BFGS');

LOGISTIC_REG

---------------------------
Finished in 20 iterations

(1 row)

See Also

- Building a Logistic Regression Model
- Regression Model Attributes
- PREDICT_LOGISTIC_REG

MSE

Returns a table that displays the mean squared error of the prediction and response columns in a linear regression model.

You cannot pass any inputs to the OVER() clause.

Syntax

MSE (target, prediction)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The response variable for the model. Must be a float.</td>
</tr>
<tr>
<td>prediction</td>
<td>The output from the PREDICT_LINEAR_REG function. If that output is saved as a table, the column containing the prediction from the function is used. Must be a float.</td>
</tr>
</tbody>
</table>

Examples

This example shows how you can execute the MSE function on an input table named faithful_testing. The response variables appear in the column obs, while the prediction
variables appear in the column pred.

```sql
=> SELECT MSE(obs, prediction) OVER()
    FROM (SELECT eruptions AS obs, PREDICT_LINEAR_REG (waiting USING PARAMETERS model_name='myLinearRegModel') AS prediction
           FROM faithful_testing) AS prediction_output;

+-----------------+-----------------------------------------------+
<table>
<thead>
<tr>
<th>mse</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.252925741352641</td>
<td>Of 110 rows, 110 were used and 0 were ignored</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>
```

**NAIVE_BAYES**

Executes the Naive Bayes algorithm on an input table or view. The result is a Naive Bayes model.

The following explains how columns are treated based on their data type:

- **FLOAT** - Values are assumed to follow some Gaussian distribution.
- **INTEGER** - Values are assumed to belong to one multinomial distribution.
- **CHAR/VARCHAR** - Values are assumed to follow some categorical distribution.
- **BOOLEAN** - Values are treated as categorical with two values.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```sql
NAIVE_BAYES ( 'model_name', 'input_relation', 'response_column', 'predictor_col1, predictor_col2, ..., predictor_coln'
               [ USING PARAMETERS exclude_columns='col1, col2, ..., coln',
                 [alpha=value] ])
```

**Arguments**

| 'model_name' | Identifies the model, where **model_name** conforms to the conventions |
It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.

**'input_relation'**

The table or view that contains the training data for building the model. If the `input_relation` is defined in Hive, you must sync the hcatalog schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function.

**'response_column'**

The name of the column in the `input_relation` that represents the dependent variable, or outcome. This column must be discrete labels that represent different class labels.

**'predictor_columns'**

A comma-separated list of the columns in the `input_relation` that...
represent the independent variables for the model.

**Supported Data Types:** INTEGER, CHAR/VARCHAR, FLOAT, BOOLEAN

Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, the function selects all the columns in `input_relation`. In this case, the `response_column` must be explicitly excluded, using the list assigned to `exclude_columns`.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exclude_columns='col1, col2, ..., coln'</code></td>
<td>(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>predictor_columns</code> argument.</td>
</tr>
</tbody>
</table>
| `alpha=value` | (Optional) The parameter used to control Laplace smoothing. Specifies use of Laplace smoothing if the event model is categorical, multinomial, or Bernoulli.  

**Default Value:** 1.0 |
Privileges

To use NAIVE_BAYES, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the NAIVE_BAYES function.

```sql
=> SELECT NAIVE_BAYES('naive_house84_model', 'house84_train', 'party', '*'
    USING PARAMETERS exclude_columns='party, id');
NAIVE_BAYES

Finished. Accepted Rows: 324 Rejected Rows: 0
(1 row)
```

See Also

- Classifying Data Using Naive Bayes
- PREDICT_NAIVE_BAYES
- PREDICT_NAIVE_BAYES_CLASSES
- Naive Bayes Model Attributes

NORMALIZE

Runs a normalization algorithm on an input table or view. The output is a view with the normalized data.

**Note:** This function differs from NORMALIZE_FIT, which creates and stores a model rather than creating a view definition. This could lead to different performance characteristics between the two algorithms.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.
## Syntax

```sql
NORMALIZE ( 'output_view', 'input_relation', 'input_columns', 'normalization_method'
    [ USING PARAMETERS [exclude_columns= 'col1, col2, ... coln']])
```

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'output_view'</td>
<td>The name of the View where you save the normalized data from the chosen input_columns.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains the data to be normalized. If the input_relation is defined in Hive, you must sync the hcatalog schema using the SYNC_ WITH_ HCATALOG_ SCHEMA function, and then run the machine learning function.</td>
</tr>
<tr>
<td>'input_columns'</td>
<td>A comma-</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>exclude_columns='col1, col2, ... coln'</code></td>
<td>(Optional) The columns from input_...relation...containing the values to be normalized. This argument only accepts numerical data types. To include all of the columns from the input_relation, specify this argument as a wildcard (*) parameter.</td>
</tr>
<tr>
<td>'normalization_method'</td>
<td>The normalization method to use.</td>
</tr>
<tr>
<td><strong>Valid Values:</strong></td>
<td>minmax, zscore, robust_zscore</td>
</tr>
</tbody>
</table>

If infinity values appear in the table, the method ignores those values.
Privileges

To use NORMALIZE, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view.

See GRANT (Schema) and GRANT (Table).

Examples

These examples show how you can use the NORMALIZE function on the wt and hp columns in the mtcars table.

Execute the NORMALIZE function, and specify the minmax method:

```sql
=> SELECT NORMALIZE('mtcars_norm', 'mtcars', 'wt, hp', 'minmax');

NORMALIZE

-------------------------
Finished in 1 iteration

(1 row)
```

Execute the NORMALIZE function, and specify the zscore method:

```sql
=> SELECT NORMALIZE('mtcars_normz','mtcars', 'wt, hp', 'zscore');

NORMALIZE

-------------------------
Finished in 1 iteration

(1 row)
```

Execute the NORMALIZE function, and specify the robust_zscore method:

```sql
=> SELECT NORMALIZE('mtcars_normz', 'mtcars', 'wt, hp', 'robust_zscore');

NORMALIZE

-------------------------
Finished in 1 iteration

(1 row)
```
See Also

- Normalizing Data
- Normalization Attributes

NORMALIZE_FIT

Computes normalization parameters for each of the specified columns in an input table or view. The resulting model stores the normalization parameters. For example, for MinMax normalization, the minimum and maximum value of each column are stored in the model. The generated model will serve as input to the functions APPLY_NORMALIZE and REVERSE_NORMALIZE.

For robust_zscore, NORMALIZE_FIT uses the APPROXIMATE_MEDIAN [Aggregate] function.

This function differs from NORMALIZE, which directly outputs a view with normalized results, rather than storing normalization parameters into a model for later operation.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

\[
\text{NORMALIZE}_\text{FIT} \left( \text{'model\_name'}, \text{'input\_relation'}, \text{'input\_columns'}, \text{'normalization\_method'} \right) \left[ \text{USING PARAMETERS} \left[ \text{exclude\_columns='col1, col2, ... coln'}, \right] \left[ \text{output\_view='output\_view'} \right] \right) \]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'model_name'</td>
<td>Identifies the model, where model_name conforms to the conventions described in Identifiers. It must also be</td>
</tr>
</tbody>
</table>
unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.

| `'input_relation'` | The table or view that contains the data to be normalized. If the `input_relation` is defined in Hive, you must sync the hcatalog schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function. |
| `'input_columns'`  | A comma-separated list of the columns in `input_relation` that contains the values to be |
This argument only accepts numerical data types.
To include all of the columns from the input_relation, specify this argument as a wildcard (*).

**Parameters**

| exclude_columns='col1, col2, ... coln' | (Optional) The columns from the input_relation that you want to exclude from the input_columns argument. |

't normalization_method'

The normalization method to use.

**Valid Values:**

- minmax
- zscore
- robust_zscore

If infinity values, negative infinity values, or NULL values appear in the table, the method ignores those values.
output_view = 'output view'

(Optional) The name of the View that contains all columns from the input_relation, with the specified input_columns normalized.

**Privileges**

To use NORMALIZE_FIT, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view.

See GRANT (Schema) and GRANT (Table).

**Examples**

These examples show how you can use the NORMALIZE_FIT function on the wt and hp columns in the mtcars table.

*Note: If a column contains only one distinct value, when you use the APPLY_NORMALIZE function, it returns NaN for values in that column.*

Execute the NORMALIZE_FIT function, and specify the minmax method:

```sql
=> SELECT NORMALIZE_FIT('mtcars_normfit', 'mtcars', 'wt,hp', 'minmax');
NORMALIZE_FIT
-------------
Success
(1 row)
```

Execute the NORMALIZE_FIT function, and specify the zscore method:

```sql
=> SELECT NORMALIZE_FIT('mtcars_normfitz', 'mtcars', 'wt,hp', 'zscore');
NORMALIZE_FIT
-------------
Success
(1 row)
```

Execute the NORMALIZE_FIT function, and specify the robust_zscore method:

```sql
=> SELECT NORMALIZE_FIT('mtcars_normfitrz', 'mtcars', 'wt,hp', 'robust_zscore');
NORMALIZE_FIT
-------------
```
One Hot Encoder Fit

Generates a sorted list of each of the category levels for each feature that will be encoded and stores the model.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```sql
ONE_HOT_ENCODER_FIT ( 'model_name', 'input_relation','input_columns'
 [USING PARAMETERS
   [exclude_columns='col1, col2, ... coln',]
   [output_view='output_view',]
   [extra_levels='json_string']])
```

**Arguments**

| `'model_name'` | Identifies the model, where `model_name` conforms to the conventions described in **Identifiers**. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names |
are case-insensitive.

<table>
<thead>
<tr>
<th>'input_relation'</th>
<th>The table or view that contains the data for one hot encoding. If the input_relation is defined in Hive, you must sync the hcatalog schema using the <code>SYNC_WITH_HCATALOG_SCHEMA</code> function, and then run the machine learning function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>'input_columns'</td>
<td>A comma-separated list of the columns from input_relation to be used for one hot encoding. The input_columns argument supports the use of a wildcard (*) character in place of column names.</td>
</tr>
</tbody>
</table>

**Parameters**

| exclude_columns='col1, col2, ... col' | (Optional) The columns from input_relation which you want to exclude from the input_columns argument.  
**Default Value:** Empty |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------|
| output_view='output_view'            | (Optional) The name of the view that stores input_relation and the one hot encodings. Columns are returned in the order they appear in input_relation, with the one-hot encoded columns appended after the original columns.  
**Default Value:** Empty |
| extra_levels = 'json_string'         | (Optional) Allows the user to specify additional levels in each category that are not present in input_relation. This parameter should be passed as a JSON string with category names as keys and lists of extra |
levels in each category as values. Note that hyper parameter names and string values should be quoted using the JSON standard.

Default Value: Empty.

Privileges

To use ONE_HOT_ENCODER_FIT, you must either be a superuser or have CREATE privileges for the schema and SELECT privileges for the table.

See GRANT (Schema) and GRANT (Table).

Examples

The following example shows how you can use the ONE_HOT_ENCODER_FIT function:

```
=> SELECT ONE_HOT_ENCODER_FIT ('one_hot_encoder_model','mtcars','*'
USING PARAMETERS exclude_columns='mpg,disp,drat,wt,qsec,vs,am');
ONE_HOT_ENCODER_FIT
------------------
Success
(1 row)
```

See Also

- APPLY_ONE_HOT_ENCODER
- Encoding Categorical Columns
- One Hot Encoder Attributes

PREDICT_LINEAR_REG

Applies a linear regression model on an input table or view.
## Syntax

```
PREDICT_LINEAR_REG ( col1, col2, ... coln
                    USING PARAMETERS model_name = 'model_name'
                    [, match_by_pos = 'method'])
```

## Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

## Parameters

<table>
<thead>
<tr>
<th>model_name = 'model_name'</th>
<th>The name of the linear regression model. Model names are case-insensitive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>match_by_pos = 'method'</td>
<td>(Optional) Valid Values:</td>
</tr>
<tr>
<td></td>
<td>- false (default): Input columns will be matched to features in the</td>
</tr>
<tr>
<td></td>
<td>model based on their names.</td>
</tr>
<tr>
<td></td>
<td>- true: Input columns will be matched to features in the model</td>
</tr>
<tr>
<td></td>
<td>based on their position in the list of indicated input columns.</td>
</tr>
</tbody>
</table>

## Return

| Return data type: FLOAT | Returns the predicted value. |

## Examples

The following example shows how you can use the PREDICT_LINEAR_REG function on an input table.

```
=> SELECT PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='myLinearRegModel')FROM faithful ORDER BY id;
```
PREDICT_LINEAR_REG

4.15403481386324
2.18505296804024
3.76023844469864
2.8151271587036
4.62659045686076
2.26381224187316
4.86286827835952
4.62659045686076
1.94877514654148
4.62659045686076
2.18505296804024
.
.
.
(272 rows)

The following example shows how you can use the PREDICT_LINEAR_REG function on an input table, using the match_by_pos parameter. Note that you can replace the column argument with a constant that does not match an input column:

```sql
=> SELECT PREDICT_LINEAR_REG(55 USING PARAMETERS model_name='linear_reg_faithful',
match_by_pos='true')FROM faithful ORDER BY id;
```

PREDICT_LINEAR_REG

2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
2.28552115094171
.
.
.
(272 rows)

**PREDICT_LOGISTIC_REG**

Applies a logistic regression model on an input table or view.

**Syntax**

```sql
PREDICT_LOGISTIC_REG ( col1, col2, ... coln
                      USING PARAMETERS model_name = 'model_name'
                      [, type = 'prediction_type',]
                      [cutoff = probability_cutoff,]
                      [match_by_pos = 'method'])
```
### Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

### Parameters

| model_name = 'model_name' | The name of the logistic regression model. Model names are case-insensitive. |
| type = 'prediction_type' | (Optional) Determines the type of prediction for logistic regression.  
When response is selected the predicted values are 0 or 1.  
When probability is selected, the output will be the probability of the predicted category to be 1.  
**Valid Values**  
- response (Default Value)  
- probability |
| cutoff = probability_cutoff | (Optional). Used in conjunction with type. Valid responses are between 0 and 1, exclusive. When the value of type is "response", the returned value of prediction would be 1 if its corresponding probability is bigger than or equal to the value of cutoff; otherwise, it is 0.  
**Default Value: 0.5** |
| match_by_pos = 'method' | (Optional) **Valid Values:**  
- false (default): Input columns will be matched to features in the model based on their names.  
- true: Input columns will be matched to features in the model based on their position in the list of indicated input columns. |
Return

Return data type: FLOAT

Returns the predicted class or the probability of the predicted class, depending on the response input. The return can be cast to INTEGER or other numeric types when the return is in the probability of the predicted class.

Examples

The following example shows how you can use the PREDICT_LOGISTIC_REG function on an input table.

```sql
=> SELECT car_model,
       PREDICT_LOGISTIC_REG(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb
       USING PARAMETERS model_name='myLogisticRegModel')
FROM mtcars;
```

<table>
<thead>
<tr>
<th>car_model</th>
<th>PREDICT_LOGISTIC_REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camaro Z28</td>
<td>0</td>
</tr>
<tr>
<td>Fiat 128</td>
<td>1</td>
</tr>
<tr>
<td>Fiat X1-9</td>
<td>1</td>
</tr>
<tr>
<td>Ford Pantera L</td>
<td>1</td>
</tr>
<tr>
<td>Merc 450SE</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>0</td>
</tr>
<tr>
<td>Toyota Corona</td>
<td>0</td>
</tr>
<tr>
<td>AMC Javelin</td>
<td>0</td>
</tr>
<tr>
<td>Cadillac Fleetwood</td>
<td>0</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>1</td>
</tr>
<tr>
<td>Dodge Challenger</td>
<td>0</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>0</td>
</tr>
<tr>
<td>Lotus Europa</td>
<td>1</td>
</tr>
<tr>
<td>Merc 230</td>
<td>0</td>
</tr>
<tr>
<td>Merc 280</td>
<td>0</td>
</tr>
<tr>
<td>Merc 280C</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SLC</td>
<td>0</td>
</tr>
<tr>
<td>Pontiac Firebird</td>
<td>0</td>
</tr>
<tr>
<td>Porsche 914-2</td>
<td>1</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>1</td>
</tr>
<tr>
<td>Valiant</td>
<td>0</td>
</tr>
<tr>
<td>Chrysler Imperial</td>
<td>0</td>
</tr>
<tr>
<td>Duster 360</td>
<td>0</td>
</tr>
<tr>
<td>Ferrari Dino</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>1</td>
</tr>
<tr>
<td>Hornet Sportabout</td>
<td>0</td>
</tr>
<tr>
<td>Lincoln Continental</td>
<td>0</td>
</tr>
<tr>
<td>Maserati Bora</td>
<td>1</td>
</tr>
<tr>
<td>Mazda RX4</td>
<td>1</td>
</tr>
<tr>
<td>Mazda RX4 Wag</td>
<td>1</td>
</tr>
<tr>
<td>Merc 240D</td>
<td>0</td>
</tr>
<tr>
<td>Volvo 142E</td>
<td>1</td>
</tr>
</tbody>
</table>
The following example shows how you can use the PREDICT_LOGISTIC_REG function on an input table, using the match_by_pos parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the mpg column was replaced with the constant 20:

```sql
=> SELECT car_model,
    PREDICT_LOGISTIC_REG(20, cyl, disp, drat, wt, qsec, vs, gear, carb
    USING PARAMETERS model_name='myLogisticRegModel', match_by_pos='true')
FROM mtcars;
```

<table>
<thead>
<tr>
<th>car_model</th>
<th>PREDICT_LOGISTIC_REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC Javelin</td>
<td>0</td>
</tr>
<tr>
<td>Cadillac Fleetwood</td>
<td>0</td>
</tr>
<tr>
<td>Camaro Z28</td>
<td>0</td>
</tr>
<tr>
<td>Chrysler Imperial</td>
<td>0</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>1</td>
</tr>
<tr>
<td>Dodge Challenger</td>
<td>0</td>
</tr>
<tr>
<td>Duster 360</td>
<td>0</td>
</tr>
<tr>
<td>Ferrari Dino</td>
<td>1</td>
</tr>
<tr>
<td>Fiat 128</td>
<td>1</td>
</tr>
<tr>
<td>Fiat X1-9</td>
<td>1</td>
</tr>
<tr>
<td>Ford Pantera L</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>1</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>0</td>
</tr>
<tr>
<td>Hornet Sportabout</td>
<td>0</td>
</tr>
<tr>
<td>Lincoln Continental</td>
<td>0</td>
</tr>
<tr>
<td>Lotus Europa</td>
<td>1</td>
</tr>
<tr>
<td>Maserati Bora</td>
<td>1</td>
</tr>
<tr>
<td>Mazda RX4</td>
<td>1</td>
</tr>
<tr>
<td>Mazda RX4 Wag</td>
<td>1</td>
</tr>
<tr>
<td>Merc 230</td>
<td>0</td>
</tr>
<tr>
<td>Merc 240D</td>
<td>0</td>
</tr>
<tr>
<td>Merc 280</td>
<td>0</td>
</tr>
<tr>
<td>Merc 280C</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SE</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>0</td>
</tr>
<tr>
<td>Merc 450SLC</td>
<td>0</td>
</tr>
<tr>
<td>Pontiac Firebird</td>
<td>0</td>
</tr>
<tr>
<td>Porsche 914-2</td>
<td>1</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>1</td>
</tr>
<tr>
<td>Toyota Corona</td>
<td>0</td>
</tr>
<tr>
<td>Valiant</td>
<td>0</td>
</tr>
<tr>
<td>Volvo 142E</td>
<td>1</td>
</tr>
</tbody>
</table>

(32 rows)

**PREDICT_NAIVE_BAYES**

Applies a Naive Bayes model on an input table or view.
Syntax

PREDICT NAIVE_BAYES ( predictor_columns
    USING PARAMETERS model_name = 'model_name'
    [, type = ' { RESPONSE | PROBABILITY } ',]
    [class = 'user_input_class', ]
    [match_by_pos = 'method'] )

Arguments

predictor_columns | A comma-separated list of the columns in input_relation that represent the independent features for the model.
                   | Supports the use of wildcard (*) characters in place of column names.

Parameters

model_name = 'model_name' | The name of the naive bayes model. Model names are case-insensitive.

type = ' { RESPONSE | PROBABILITY } ' | (Optional) Specifies that the function can take the value RESPONSE or PROBABILITY. RESPONSE, using the class result of the prediction. This value uses the highest probability among all possible classes. PROBABILITY defers to the second argument 'class'.
                              | Default Value: response

class = 'user_input_class' | (Optional) Specifies a specific class to use when type is set to PROBABILITY. The predict function returns the probability of belonging to this given class as predicted by the classifier. If class is not specified, its default value is the predicted class -- the highest will be returned. Thus, the predict function returns the probability that the input instance belonging to its predicted class.

match_by_pos = 'method' | (Optional) Valid Values:
                           | • false (default): Input columns will be matched to features in the model based on their names.
Return

<table>
<thead>
<tr>
<th>Return data type:</th>
<th>VARCHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the predicted class or the probability of the predicted class, depending on the response input. The return can be cast to INTEGER or other numeric types when the return is in the probability of the predicted class.</td>
<td></td>
</tr>
</tbody>
</table>

Examples

This example shows how you can use the PREDICT_NAIVE_BAYES function.

```sql
=> SELECT party, PREDICT_NAIVE_BAYES (vote1, vote2, vote3
    USING PARAMETERS model_name = 'naive_house84_model',
    type = 'response')
AS Predicted_Party
FROM house84_test;

<table>
<thead>
<tr>
<th>party</th>
<th>Predicted_Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>republican</td>
<td>republican</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>republican</td>
<td>republican</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>republican</td>
<td>republican</td>
</tr>
<tr>
<td>democrat</td>
<td>democrat</td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>democratic</td>
<td></td>
</tr>
<tr>
<td>(99 rows)</td>
<td></td>
</tr>
</tbody>
</table>
```
See Also

- Classifying Data Using Naive Bayes
- NAIVE_BAYES
- PREDICT NAIVE_BAYES CLASSES

PREDICT NAIVE_BAYES CLASSES

Applies a Naive Bayes model on an input table or view and returns the probabilities of classes.

Syntax

```
PREDICT NAIVE_BAYES CLASSES (predictor_columns
    USING PARAMETERS model_name = 'model_name'
    [,key_columns = 'key_columns',]
    [exclude_columns = 'col1, col2, ..., coln',]
    [classes = 'class1, class2, ..., classn', ]
    [match_by_pos = 'method' ] )
OVER()
    AS (key_columns, Predicted, Probability, class1, class2, ..., classn)
```

Arguments

| predictor_columns | A comma-separated list of the columns in input_relation that represent the independent variables for the model. Supports the use of wildcard (*) characters in place of column names. If you use a wildcard character (*) in place of a column name, all the columns in input_relation are selected. |

Parameters

| model_name = 'model_name' | The name of the naive bayes model. Model names are case insensitive. |
| key_columns = 'key_columns' | (Optional) A comma-separated list of column names from the input_relation which you use |
to identify each row of the output. Columns specified in `key_columns` should be also part of `predictor_columns`. To exclude any columns (including the columns specified in `key_columns`) from being used for prediction, they need to be listed in `exclude_columns`.

```
exclude_columns = 'col1, col2, ..., coln'
```

(Optional) The columns from `predictor_columns` that you want to exclude. This parameter is useful when using the wildcard (*) in the `predictor_columns`.

```
classes = 'class1, class2, ..., classn'
```

(Optional) Class labels in the model. The probability of belonging to this given class as predicted by the classifier. The values are case sensitive.

```
match_by_pos= 'method'
```

(Optional) Valid Values:

- **false** (default): Input columns will be matched to features in the model based on their names.
- **true**: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

## Return

<table>
<thead>
<tr>
<th>Return data type: One VARCHAR column and multiple FLOAT columns</th>
<th>The VARCHAR column is named predicted and contains the class label with the highest probability. The first FLOAT column is named probability and contains the probability for the class</th>
</tr>
</thead>
</table>
specified in the predicted column. The other FLOAT columns contain each class listed in the classes column.

Examples

This example shows how you can use the PREDICT_NAIVE_BAYES_CLASSES function.

```sql
-> SELECT PREDICT_NAIVE_BAYES_CLASSES (id, vote1, vote2
  USING PARAMETERS model_name = 'naive_house84_model',
  key_columns = 'id',
  exclude_columns = 'id',
  classes = 'democrat, republican',
  match_by_pos = 'false')
OVER() FROM house84_test;
```

<table>
<thead>
<tr>
<th>id</th>
<th>Predicted</th>
<th>Probability</th>
<th>democrat</th>
<th>republican</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>democrat</td>
<td>0.775473383353576</td>
<td>0.775473383353576</td>
<td>0.224526616646424</td>
</tr>
<tr>
<td>28</td>
<td>democrat</td>
<td>0.775473383353576</td>
<td>0.775473383353576</td>
<td>0.224526616646424</td>
</tr>
<tr>
<td>83</td>
<td>republican</td>
<td>0.592510497724379</td>
<td>0.407489502275621</td>
<td>0.592510497724379</td>
</tr>
<tr>
<td>102</td>
<td>democrat</td>
<td>0.779889432167111</td>
<td>0.779889432167111</td>
<td>0.220110567832889</td>
</tr>
<tr>
<td>107</td>
<td>republican</td>
<td>0.598662714551597</td>
<td>0.401337285448403</td>
<td>0.598662714551597</td>
</tr>
<tr>
<td>125</td>
<td>republican</td>
<td>0.598662714551597</td>
<td>0.401337285448403</td>
<td>0.598662714551597</td>
</tr>
<tr>
<td>132</td>
<td>republican</td>
<td>0.592510497724379</td>
<td>0.407489502275621</td>
<td>0.592510497724379</td>
</tr>
<tr>
<td>136</td>
<td>republican</td>
<td>0.592510497724379</td>
<td>0.407489502275621</td>
<td>0.592510497724379</td>
</tr>
<tr>
<td>155</td>
<td>republican</td>
<td>0.598662714551597</td>
<td>0.401337285448403</td>
<td>0.598662714551597</td>
</tr>
<tr>
<td>174</td>
<td>republican</td>
<td>0.592510497724379</td>
<td>0.407489502275621</td>
<td>0.592510497724379</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- Classifying Data Using Naive Bayes
- PREDICT_NAIVE_BAYES
- NAIVE_BAYES
PREDICT_RF_CLASSIFIER_CLASSES

Applies a random forest model on an input table or view and returns the probabilities of classes. The predicted class is selected only based on the popular vote of the decision trees in the forest. Therefore, in special cases the calculated probability of the predicted class may not be the highest.

Syntax

PREDICT_RF_CLASSIFIER_CLASSES ( col1, col2, ... coln

USING PARAMETERS model_name = 'model_name'
    [key_columns = 'key_columns',]
    [exclude_columns = 'col1, col2, ..., coln',]
    [classes = 'class1, class2, ..., classn', ]
    [match_by_pos = 'method' ]

OVER([window-partition-clause])

Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

Parameters

<table>
<thead>
<tr>
<th>model_name = 'model_name'</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name of the random forest model. Model names are case-insensitive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key_columns = 'key_columns'</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Optional) A comma-separated list of column names from input_relation which you use to identify each row of the output. Columns specified in key_columns should be also part of predictor_columns. To exclude any columns (including the columns specified in key_columns) from being used for prediction, they need to be listed in exclude_columns.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>exclude_columns = 'col1, col2, ..., coln'</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Optional) The columns from predictor_columns that you want to exclude. This parameter is useful when using the wildcard (*) in the predictor_columns.</td>
</tr>
</tbody>
</table>
classes = 'class1, class2, ..., classn'

(Optional) Class labels in the model. The probability of belonging to this given class as predicted by the classifier. The values are case sensitive.

match_by_pos = 'method'

(Optional) Valid Values:

- false (default): Input columns will be matched to features in the model based on their names.
- true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

Return

Return data type: One VARCHAR column and multiple FLOAT columns

The predicted column, of type VARCHAR, contains the class label with the highest vote (popular vote). The first FLOAT column is named probability and contains the probability for the class reported in the predicted column. The other FLOAT columns contain the probability of each class specified in the classes input parameter.

Key columns

Columns with the same value and data type as the matching input columns that are specified in the key_columns input parameter.

Examples

This example shows how you can use the PREDICT_RF_CLASSIFIER_CLASSES function.

```
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
USING PARAMETERS model_name='myRFModel') OVER () FROM iris;
```
This example shows how you can use the PREDICT_RF_CLASSIFIER_CLASSES function, using the match_by_pos parameter:

```sql
=> SELECT PREDICT_RF_CLASSIFIER_CLASSES(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
   USING PARAMETERS model_name='myRFModel', match_by_pos='true') OVER () FROM iris;
```

<table>
<thead>
<tr>
<th>predicted</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>setosa</td>
<td>0.99</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
</tr>
<tr>
<td>(150 rows)</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- [Classifying Data Using Random Forest](#)
- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER](#)
- [GET_MODEL_SUMMARY](#)

**PREDICT_RF_CLASSIFIER**

Applies a random forest model on an input table or view. The predicted class is selected only based on the popular vote of the decision trees in the forest. Therefore, in special cases the calculated probability of the predicted class may not be the highest.
### Syntax

```
PREDICT_RF_CLASSIFIER ( col1, col2, ... coln
    USING PARAMETERS model_name = 'model_name'
        [, type= ' { RESPONSE | PROBABILITY }',]
            [class= 'user_input_class',]
        [match_by_pos = 'method'])
```

### Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name = 'model_name'</td>
<td>The name of the random forest model. Model names are case-insensitive.</td>
</tr>
</tbody>
</table>
| type= ' { RESPONSE | PROBABILITY }' | (Optional) Determines the type of prediction for random forest. When response is selected the result of the prediction is the class with the highest probability among all possible classes. When probability is selected, the 'class' parameter is considered. **Valid Values**
  - response (Default Value)
  - probability |
| class = 'user_input_class' | (Optional) Specifies a specific class to use when type is set to PROBABILITY. The predict function returns the probability of the specified class to be the response. If class is not specified, its default value is the predicted class -- the one with popular vote. Thus, the predict function returns the probability that the input instance belonging to its predicted class. **Default Value:** Auto |
**match_by_pos= 'method'**

(Optional) Valid Values:

- **false** (default): Input columns will be matched to features in the model based on their names.
- **true**: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

**Return**

| Return data type: VARCHAR | The predict function returns the predicted class (based on popular votes) or probability of a class (depending on the value of the optional input parameters type and class) for each input instance. |

**Examples**

This example shows how you can use the `PREDICT_RF_CLASSIFIER` function.

```
=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
      USING PARAMETERS model_name='myRFModel') FROM iris;
```

```
PREDICT_RF_CLASSIFIER
-----------------------
setosa
setosa
setosa

(150 rows)
```

This example shows how you can use the `PREDICT_RF_CLASSIFIER` function, using the `match_by_pos` parameter:
=> SELECT PREDICT_RF_CLASSIFIER (Sepal_Length, Sepal_Width, Petal_Length, Petal_Width
   USING PARAMETERS model_name='myRFModel', match_by_pos='true') FROM iris;

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>setosa</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>versicolor</td>
<td>versicolor</td>
<td>versicolor</td>
<td>versicolor</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>virginica</td>
<td>virginica</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

(150 rows)

See Also

- [Classifying Data Using Random Forest](#)
- [RF_CLASSIFIER](#)
- [PREDICT_RF_CLASSIFIER_CLASSES](#)
- [GET_MODEL_SUMMARY](#)

**PREDICT_RF_REGRESSOR**

Applies a random forest model on an input table or view.

**Syntax**

```
PREDICT_RF_REGRESSOR ( col1, col2, ... coln
   USING PARAMETERS model_name = 'model_name'
       [,match_by_pos = 'method']]  
```
Arguments

| col1, col2, ..., coln | The columns to use from the input table or view. |

Parameters

| model_name = 'model_name' | The name of the random forest model. Model names are case-insensitive. |
| match_by_pos= 'method' | (Optional) Valid Values:
  - false (default): Input columns will be matched to features in the model based on their names.
  - true: Input columns will be matched to features in the model based on their position in the list of indicated input columns. |

Return

| Return data type: FLOAT | The predict function returns the predicted value of the random forest model, which is the average of the prediction of the trees in the forest. |

Examples

This example shows how you can use the PREDICT_RF_REGRESSOR function.

```sql
=> SELECT PREDICT_RF_REGRESSOR (mpg,cyl,hp,drat,wt
USING PARAMETERS model_name='myRFRegressorModel')FROM mtcars;
```

```
2.94774203574204
2.6954087024087
2.6954087024087
2.89906346431346
2.97688489288489
2.97688489288489
2.7086587024087
2.92078965478965
2.97688489288489
2.7086587024087
```
See Also

- Building a Random Forest for Regression Model
- GET_MODEL_SUMMARY
- RF_REGRESSOR

**PREDICT_SVM_CLASSIFIER**

Applies an SVM model on an input table or view.

**Syntax**

```
PREDICT_SVM_CLASSIFIER (input_columns
     USING PARAMETERS model_name='model_name'
     [, match_by_pos = 'method'])
```

**Arguments**

| input_columns | A comma-separated list of the columns to be used for prediction. |
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>model_name='model_name'</code></td>
<td>The name of the SVM classifier model. Model names are case-insensitive.</td>
</tr>
<tr>
<td><code>match_by_pos= 'method'</code></td>
<td>(Optional) Valid Values:</td>
</tr>
<tr>
<td></td>
<td>• false (default): Input columns will be matched to features in the model based on their names.</td>
</tr>
<tr>
<td></td>
<td>• true: Input columns will be matched to features in the model based on their position in the list of indicated input columns.</td>
</tr>
</tbody>
</table>

Return

| Return data type: FLOAT | Returns the predicted value. |

Examples

This example shows how you can use the `PREDICT_SVM_CLASSIFIER` function on the `mtcars` table:

```sql
=> SELECT PREDICT_SVM_CLASSIFIER (mpg,cyl,disp,wt,qsec,vs,gear,carb US
USING PARAMETERS model_name='mySvmClassModel') FROM mtcars;
```

```
PREDICT_SVM_CLASSIFIER
------------------------
0
0
1
0
0
1
1
1
1
0
0
1
0
0
1
```
This example shows how you can use the PREDICT_SVM_CLASSIFIER function on the mtcars table, using the match_by_pos parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the mpg column was replaced with the constant 40:

```sql
=> SELECT PREDICT_SVM_CLASSIFIER (40,cyl,disp,wt,qsec,vs,gear,carb
USING PARAMETERS model_name='mySvmClassModel', match_by_pos = 'true') FROM mtcars;
```

```
PREDICT_SVM_CLASSIFIER
------------------------
0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
```
See Also

- Classifying Data Using SVM (Support Vector Machine)
- SVM (Support Vector Machine) for Classification
- SVM_CLASSIFIER
- GET_MODEL_SUMMARY

PREDICT_SVM_REGRESSOR

Applies an SVM model on an input table or view.

Syntax

PREDICT_SVM_REGRESSOR(input_columns
                     USING PARAMETERS model_name='model_name'
                     [, match_by_pos = 'method'])

Arguments

| input_columns | A comma-separated list of the columns to be used for prediction. |

Parameters

<table>
<thead>
<tr>
<th>model_name='model_name'</th>
<th>The name of the SVM regressor model. Model names are case-insensitive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>match_by_pos='method'</td>
<td>(Optional) Valid Values:</td>
</tr>
</tbody>
</table>
- **false** (default): Input columns will be matched to features in the model based on their names.
- **true**: Input columns will be matched to features in the model based on their position in the list of indicated input columns.

## Return

Return data type: FLOAT

Returns the predicted value.

## Examples

This example shows how you can use the `PREDICT_SVM_REGRESSOR` function on the faithful table:

```sql
=> SELECT PREDICT_SVM_REGRESSOR(waiting USING PARAMETERS model_name='mySvmRegModel',
                                  match_by_pos = 'true')
               FROM faithful ORDER BY id;
```

```
PREDICT_SVM_REGRESSOR
----------------------
 4.06488248694445
 2.30392277646291
 3.71269054484815
 2.867429883817
 4.48751281746003
 2.37436116488217
 4.69882798271781
 4.48751281746003
 2.09260761120512
...
(272 rows)
```

This example shows how you can use the `PREDICT_SVM_REGRESSOR` function on the faithful table, using the `match_by_pos` parameter. Note that you can any of the column inputs with a constant that does not match an input column. In this example, the waiting column was replaced with the constant 40:

```sql
=> SELECT PREDICT_SVM_REGRESSOR(40 USING PARAMETERS model_name='mySvmRegModel',
                                 match_by_pos = 'true')
              FROM faithful ORDER BY id;
```

```
PREDICT_SVM_REGRESSOR
----------------------
 1.31778533859324
```
See Also

- Building an SVM for Regression Model
- SVM (Support Vector Machine) for Regression
- SVM_REGRESSOR
- GET_MODEL_SUMMARY

REVERSE_NORMALIZE

Reverses the normalization transformation on normalized data, thereby de-normalizing the normalized data. If normalized data is used as the input, the original data is returned. If you specify a column that is not in the specified model, REVERSE_NORMALIZE returns that column unchanged.

Syntax

```
REVERSE_NORMALIZE ( col1, col2 ... , coln
                     USING PARAMETERS model_name='model_name');
```

Arguments

- `col1, col2 ... , coln`: The columns to use from the input table or view.
Parameters

| model_name='model_name' | The name of the model. Model names are case-insensitive. Value must be VARCHAR. |

Privileges

To use REVERSE_NORMALIZE, you must either be the dbadmin, owner of the model or have USAGE privileges. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the REVERSE_NORMALIZE function on the hp and cyl columns in the mtcars table, where hp is in the normalization model and cyl is not in the normalization model.

```sql
=> SELECT REVERSE_NORMALIZE (hp, cyl USING PARAMETERS model_name = 'mtcars_normfit') FROM mtcars;
hp   | cyl
-----+-----
42502| 8
58067| 8
26371| 4
42502| 8
31182| 6
32031| 4
26937| 4
34861| 6
34861| 6
50992| 8
50992| 8
49577| 8
25805| 4
18447| 4
29767| 6
65142| 8
69387| 8
14768| 4
49577| 8
60897| 8
94857| 8
31182| 6
31182| 6
30899| 4
69387| 8
```
See Also

- APPLY_NORMALIZE
- NORMALIZE
- NORMALIZE_FIT
- Normalizing Data

**RF_CLASSIFIER**

Trains a random forest model for classification on an input table or view.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```sql
RF_CLASSIFIER ( 'model_name', 'input_relation', 'response_column', 'predictor_col1, predictor_col2, ..., predictor_coln'

[USING PARAMETERS [exclude_columns= 'col1, col2, ...,coln',]
 [ntree= value,]
 [mtry= value,]
 [sampling_size= value,]
 [max_depth= value,]
 [max_breadth= value,]
 [min_leaf_size= value,]
 [min_info_gain= value,]
 [nbins= value] ])
```
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'model_name'</td>
<td>The name of the model stored as a result of the training. Identifies the model, where <em>model_name</em> conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains the training samples. If the input_relation is defined in Hive, you must sync the hcatalog schema using the <code>SYNC_WITH_HCATALOG_SCHEMA</code> function, and then run the machine learning function.</td>
</tr>
<tr>
<td>'response_column'</td>
<td>The name of the column in input_relation that represents the</td>
</tr>
<tr>
<td>dependent variable. This column must be of data type CHAR or VARCHAR.</td>
<td></td>
</tr>
<tr>
<td>'predictor_columns' A comma-separated list of the columns in the input_relation that represent the independent variables for the model. These columns must be of CHAR, VARCHAR, BOOLEAN, INTEGER and FLOAT data types. CHAR, VARCHAR and BOOLEAN are treated as categorical data types. All other data types are treated as numeric data types.</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<p>| exclude_columns='col1, col2, ..., coln' (Optional) The columns from input_relation that you want to exclude from the predictor_columns argument. |
| ntree=value (Optional) A positive integer number that indicates the number of trees in the forest. Default Value: 20 Valid Range: [0 to 1000] |
| mtry=value (Optional) A positive integer number that indicates... |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mtry</code></td>
<td>the number of features to be considered at the split of a tree node. <strong>Default Value:</strong> When no value is specified for <code>mtry</code>, its default value is the square root of the total number of predictors. <strong>Valid Range:</strong> A positive integer number, smaller than or equal to the number of predictors.</td>
</tr>
<tr>
<td><code>sampling_size=value</code></td>
<td><em>(Optional)</em> A number that indicates what portion of the input data set will randomly be picked for training each tree. <strong>Default Value:</strong> 0.632 <strong>Valid Range:</strong> <em>(0.0,1.0)</em></td>
</tr>
<tr>
<td><code>max_depth=value</code></td>
<td><em>(Optional)</em> A positive integer number that specifies the maximum depth for growing each tree. <strong>Default Value:</strong> 5 <strong>Valid Range:</strong> <em>[1 to 100]</em></td>
</tr>
<tr>
<td><code>max_breadth=value</code></td>
<td><em>(Optional)</em> A positive integer number that specifies the maximum number of leaf nodes a tree in the forest can have. <strong>Default Value:</strong> 32 <strong>Valid Range:</strong> <em>[1 to 1e9]</em></td>
</tr>
<tr>
<td><code>min_leaf_size=value</code></td>
<td><em>(Optional)</em> A positive integer number that specifies the minimum samples each branch must have after splitting a node. A split that causes fewer remaining samples will be discarded. <strong>Default Value:</strong> 1 <strong>Valid Range:</strong> <em>[1 to 1e6]</em></td>
</tr>
<tr>
<td><code>min_info_gain=value</code></td>
<td><em>(Optional)</em> A non-negative number. Any split with information gain less than this threshold will be discarded. <strong>Default Value:</strong> 0.0 <strong>Valid Range:</strong> <em>[0.0 to 1.0]</em></td>
</tr>
</tbody>
</table>
**nbins=value**

(Optional) A positive integer number that indicates the number of bins to use for continuous features.

Default Value: 32

Valid Range: [2 to 1000]

---

**Privileges**

To use RF_CLASSIFIER, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

---

**Examples**

This example shows how you can use the RF_CLASSIFIER function.

```sql
=> SELECT RF_CLASSIFIER ('myRFModel', 'iris', 'Species', 'Sepal_Length, Sepal_Width, Petal_Length, Petal_Width'
USING PARAMETERS ntree=100, sampling_size=0.3);
RF_CLASSIFIER
------------------------------------------------------------------------
The random forest is trained
(1 row)
```

---

**See Also**

- [Classifying Data Using Random Forest](#)
- **PREDICT_RF_CLASSIFIER**
- **PREDICT_RF_CLASSIFIER_CLASSES**
- **GET_MODEL_SUMMARY**

---

**RF_REGRESSOR**

Trains a random forest model for regression on an input table or view.
Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```sql
RF_REGRESSOR ( 'model_name', 'input_relation', 'response_column', 'predictor_col1, predictor_col2, ..., predictor_coln'

[USING PARAMETERS [exclude_columns= 'col1, col2, ..., coln',]
 [ntree= value,]
 [mtry= value,]
 [sampling_size= value,]
 [max_depth= value,]
 [max_breadth= value,]
 [min_leaf_size= value,]
 [min_info_gain= value,]
 [nbins= value] ])
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'model_name'</code></td>
<td>The name of the model stored as a result of the training. Identifies the model, where model_name conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.</td>
</tr>
<tr>
<td><code>'input_relation'</code></td>
<td>The table or view that contains the training samples. If</td>
</tr>
</tbody>
</table>
the input_relation is defined in Hive, you must sync the hcatalog schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function.

| 'response_column' | The name of the column in input_relation that represents the dependent variable. This column must be of data type FLOAT or INTEGER. |
| 'predictor_columns' | A comma-separated list of the columns in the input_relation that represent the independent variables for the model. These columns must be of CHAR, VARCHAR, BOOLEAN, INTEGER or FLOAT data types. CHAR, VARCHAR and BOOLEAN are treated as categorical data types. All other data types are treated as numeric data types. |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exclude_columns='col1, col2, ..., coln'</code></td>
<td>(Optional) The columns from input_relation that you want to exclude from the predictor_columns argument.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ntree=value</code></td>
<td>(Optional) A positive integer number that indicates the number of trees in the forest.</td>
<td>20</td>
<td>[0, 1000]</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> When no value is specified for mtry, its default value is one-third of the total number of predictors</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Valid Range:</strong> A positive integer number, smaller than or equal to the number of predictors.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>mtry=value</code></td>
<td>(Optional) A positive integer number that indicates the number of features to be considered at the split of a tree node.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> When no value is specified for mtry, its default value is one-third of the total number of predictors</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Valid Range:</strong> A positive integer number, smaller than or equal to the number of predictors.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sampling_size=value</code></td>
<td>(Optional) A number that indicates what portion of the input data set will randomly be picked for training each tree</td>
<td>0.632</td>
<td>(0.0,1.0]</td>
</tr>
<tr>
<td><code>max_depth=value</code></td>
<td>(Optional) A positive integer number that specifies the maximum depth for growing each tree.</td>
<td>5</td>
<td>[1, 100]</td>
</tr>
<tr>
<td><code>max_breadth=value</code></td>
<td>(Optional) A positive integer number that specifies the maximum number of leaf nodes a tree in the forest can have.</td>
<td>32</td>
<td>[1, 1e9]</td>
</tr>
</tbody>
</table>
### min_leaf_size=value

(Optional) A positive integer number that specifies the minimum samples each branch must have after splitting a node. A split that causes fewer remaining samples will be discarded.

**Default Value:** 5  
**Valid Range:** [1, 1e6]

### min_info_gain=value

(Optional) A non-negative float number. Any split with information gain less than this threshold will be discarded.

**Default Value:** 0.0  
**Valid Range:** [0.0, 1.0)

### nbins=value

(Optional) A positive integer number that indicates the number of bins to use for continuous features.

**Default Value:** 32  
**Valid Range:** [2, 1000]

#### Privileges

To use `RF_REGRESSOR`, you must have CREATE privileges for the schema of the output model and SELECT privileges for the input_relation.

See `GRANT (Schema)` and `GRANT (Table)`.

#### Examples

This example shows how you can use the `RF_REGRESSOR` function.

```
=> SELECT RF_REGRESSOR ('myRFRegressorModel', 'mtcars', 'carb', 'mpg, cyl, hp, drat, wt' USING PARAMETERS ntree=100, sampling_size=0.3));
RF_REGRESSOR
-----------
Finished
(1 row)
```
See Also

- Building a Random Forest for Regression Model
- GET_MODEL_SUMMARY
- PREDICT_RF_REGRESSOR

ROC

Returns a table that displays the points on a receiver operating characteristic curve. The ROC function tells you the accuracy of a classification model as you raise the discrimination threshold for the model.

You cannot pass any inputs to the OVER() clause.

Syntax

ROC (target, probabilities
     [USING PARAMETERS [num_bins=nBins] ])
     OVER()

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The column in the input table containing the response variable. Must be an integer. Greater values result in more precise approximations of the AUC.</td>
</tr>
<tr>
<td>probabilities</td>
<td>The column in the input table where the observation is of class 1. Must be a float.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_bins=nBins</td>
<td>(Optional) Groups rows together, based upon the probability column, for faster processing. You use this parameter to determine the number of different decision boundaries to consider. The parameter partitions the number line from 0 to 1 in nBin points that are equally spaced. It evaluates</td>
</tr>
</tbody>
</table>
the table at each of the nBin points. Must be an integer.

Default Value: 100

Examples

This example show how you can execute the ROC function on an input table named mtcars. The response variables appear in the column obs, while the prediction variables appear in the column pred.

```sql
=> SELECT ROC(obs::int, prob::float USING PARAMETERS num_bins=2) OVER()
  FROM (SELECT am AS obs,
         PREDICT_LOGISTIC_REG (mpg, cyl, disp, drat, wt,
                                qsec, vs, gear, carb
         USING PARAMETERS model_name='myLogisticRegModel',
                                type='probability')
         AS prob
  FROM mtcars) AS prediction_output;
```

<table>
<thead>
<tr>
<th>decision_boundary</th>
<th>false_positive_rate</th>
<th>true_positive_rate</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Of 32 rows, 32 were used and 0 were ignored (3 rows)</td>
</tr>
</tbody>
</table>

The first column, decision_boundary, indicates the cut-off point for whether to classify a response as 0 or 1. For instance, in each row, if prob is greater than decision_boundary, the response is classified as 1. If prob is equal to or less than decision_boundary, the response is classified as 0.

The second column, false_positive_rate, shows the percentage of false positives (when 0 is classified as 1) in the corresponding decision_boundary.

The third column, true_positive_rate, shows the percentage of rows that were classified as 1 and also belong to class 1.

RSQUARED

Returns a table with the R-squared value of the predictions in a linear regression model.

Syntax

```sql
RSQUARED ( target, prediction )
OVER()
```
Important: You cannot pass any inputs to the OVER() clause.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The response variable for the model. Must be a float.</td>
</tr>
<tr>
<td>prediction</td>
<td>The output from the PREDICT_LINEAR_REG function. If that output is saved as a table, the column containing the prediction from the function is used. Must be a float.</td>
</tr>
</tbody>
</table>

Examples

This example shows how you can execute the RSQUARED function on an input table named faithful_testing. The response variables appear in the column, obs, while the prediction variables appear in the column, pred.

```sql
=> SELECT RSQUARED(obs, prediction) OVER()
FROM (SELECT eruptions AS obs, PREDICT_LINEAR_REG (waiting
    USING PARAMETERS model_name='myLinearRegModel') AS prediction
FROM faithful_testing) AS prediction_output;
```

```
<table>
<thead>
<tr>
<th>rsq</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.801392981147911</td>
<td>Of 110 rows, 110 were used and 0 were ignored</td>
</tr>
</tbody>
</table>
```

SUMMARIZE_CATCOL

Returns a statistical summary of categorical data input, such as INTEGER, BOOLEAN or string, and reports the frequency of the category. You cannot use this function on multiple columns.

Syntax

```
SUMMARIZE_CATCOL (col
    [USING PARAMETERS [TOPK = value,]
     [WITH_TOTALCOUNT = method]])
OVER() FROM table;
```
Arguments

<table>
<thead>
<tr>
<th>col</th>
<th>The name of the column you want to summarize.</th>
</tr>
</thead>
</table>

Parameters

<table>
<thead>
<tr>
<th>TOPK = value</th>
<th>(Optional) Limits the output to the most frequent rows.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WITH_ TOTALCOUNT = method</td>
<td>(Optional) Valid Values:</td>
</tr>
<tr>
<td></td>
<td>• true (default value): The first row of the output table displays a count equal to the number of rows displayed in that column and a percent equal to 100.</td>
</tr>
<tr>
<td></td>
<td>• false: The first row is not displayed.</td>
</tr>
</tbody>
</table>

Examples

This example shows the categorical summary for the current_salary column in the salary_data table. The output of the query shows the column category, count, and percent. The first column gives the categorical levels, with the same SQL data type as the input column, the second column gives a count of that value, and the third column gives a percentage.

```
=> SELECT SUMMARIZE_CATCOL (current_salary USING PARAMETERS TOPK = 5) OVER() FROM salary_data;
CATEGORY | COUNT | PERCENT
----------|-------|--------
39004 | 2 | 0.2
35321 | 1 | 0.1
36313 | 1 | 0.1
36538 | 1 | 0.1
36562 | 1 | 0.1
(6 rows)
```

SUMMARIZE_NUMCOL

Returns a statistical summary of numerical column(s) in Vertica, including count, average, standard deviation, min and max values, percentile information, and the median.
Syntax

`SUMMARIZE_NUMCOL (col1, col2, col3..., coln)
    [USING PARAMETERS [exclude_columns = 'col1, col2, ..., coln']]})
    OVER() FROM table;

Arguments

| col1, col2, col3..., coln | The name of the column(s) you want to summarize. |

Parameters

| exclude_columns='col1, col2, ... coln' | (Optional) The columns you want to exclude from the input columns argument. |

Examples

Create the employee table and insert values into the table:

```sql
=> CREATE TABLE employee (id INT PRIMARY KEY, name VARCHAR(64), age INT, gender CHAR(1), title VARCHAR(64), salary MONEY);
=> INSERT INTO employee VALUES(1, 'Leonardo da Vinci', 44, 'M', 'Artist', 1234.56);
=> INSERT INTO employee VALUES(2, 'Albert Einstein', 45, 'M', 'Scientist', 2345.67);
=> INSERT INTO employee VALUES(3, 'Myrddin Wyltt', 67, 'M', 'Wizard', 3456.78);
=> INSERT INTO employee VALUES(4, 'George Walker Bush', 71, 'M', 'President', 4567.89);
=> INSERT INTO employee VALUES(6, 'Elizabeth Alexandra Mary', 90, 'F', 'Queen', 5678.90);
```

This example shows the statistical summary for the age and salary columns in the employee table:

```sql
=> SELECT SUMMARIZE_NUMCOL('* USING PARAMETERS exclude_columns='id,name,gender,title') OVER() FROM employee;

<table>
<thead>
<tr>
<th>COLUMNS</th>
<th>COUNT</th>
<th>MEAN</th>
<th>STDDEV</th>
<th>MIN</th>
<th>PERC25</th>
<th>MEDIAN</th>
<th>PERC75</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>5</td>
<td>63.4</td>
<td>19.3209730603818</td>
<td>44</td>
<td>45</td>
<td>67</td>
<td>71</td>
</tr>
<tr>
<td>90</td>
<td>5</td>
<td>3456.76</td>
<td>1756.78754300285</td>
<td>1234.56</td>
<td>2345.67</td>
<td>3456.78</td>
<td>4567.89</td>
</tr>
<tr>
<td>5678.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)
```
SVM_CLASSIFIER

Trains the SVM model on an input table or view. You can view the model using `GET_MODEL_SUMMARY`.

**Important:** Before using a machine learning function, be aware that all the ongoing transactions might be committed.

**Syntax**

```
SVM_CLASSIFIER ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'
    [USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
    [C='value',]
    [epsilon='value',]
    [max_iterations='value'] ] )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'model_name'</td>
<td>Identifies the model, where <code>model_name</code> conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.</td>
</tr>
<tr>
<td><code>input_relation</code></td>
<td>The table or view that contains the training data. If the <code>input_relation</code> is defined in Hive, you must sync the hcatalog schema using the <code>SYNC_WITH_HCATALOG_SCHEMA</code> function, and then run the machine learning function.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `response_column` | The name of the column in `input_relation` that represents the dependent variable, or outcome. The value must be 0 or 1. **Valid Types:**  
  - FLOAT  
  - INTEGER  
  - NUMERIC |
| `predictor_columns` | A comma-
separated list of the columns in `input_relation` that represent the independent variables for the model.

**Valid Types:**
- INTEGER
- FLOAT
- NUMERIC

If the column name contains special characters, it must use double quotes.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exclude_columns='col1, col2, ... coln'</code></td>
<td>(Optional) The columns from <code>input_relation</code> that you want to exclude from the <code>input_columns</code> argument.</td>
<td>Empty</td>
</tr>
<tr>
<td><code>C='value'</code></td>
<td>(Optional) Sets the weight for misclassification cost. The algorithm minimizes the regularization cost and the misclassification cost.</td>
<td>1.0</td>
</tr>
<tr>
<td><code>epsilon='value'</code></td>
<td>(Optional) Used to control accuracy.</td>
<td>1e-3</td>
</tr>
</tbody>
</table>
max_iterations='value'

(Optional) Determines the maximum number of iterations that the algorithm performs before achieving the specified accuracy result.
Default Value: 100

Privileges

To use SVM_CLASSIFIER, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the SVM_CLASSIFIER function on the mtcars table:

```sql
=> SELECT SVM_CLASSIFIER('mySvmClassModel', 'mtcars', 'am',
'mpg,cyl,disp,drat,wt,qsec,vs,gear,carb'
USING PARAMETERS exclude_columns = 'hp,drat');

SVM_CLASSIFIER
-----------------------------------------------
Finished in 15 iterations.
Accepted Rows: 32  Rejected Rows: 0
(1 row)
```

See Also

- Classifying Data Using SVM (Support Vector Machine)
- SVM (Support Vector Machine) for Classification
- PREDICT_SVM_CLASSIFIER
- GET_MODEL_SUMMARY
- SVM Model Attributes
SVM_REGRESSOR

Trains the SVM model on an input table or view. You can view the model using GET_MODEL_SUMMARY.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

```sql
SVM_REGRESSOR ( 'model_name', 'input_relation', 'response_column', 'predictor_columns'
    [USING PARAMETERS [exclude_columns='col1, col2, ... coln',]
    [error_tolerance = value,]
    [C=value,]
    [epsilon= value,]
    [max_iterations= value] ]
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'model_name'</td>
<td>Identifies the model, where <code>model_name</code> conforms to the conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. Model names are case-insensitive.</td>
</tr>
<tr>
<td>'input_relation'</td>
<td>The table or view that contains the training data. If the input_relation is defined in Hive, you must</td>
</tr>
</tbody>
</table>
sync the hcatalog schema using the `SYNC_WITH_HCATALOG_SCHEMA` function, and then run the machine learning function.

<table>
<thead>
<tr>
<th>'response_column'</th>
<th>The name of the column in <code>input_relation</code> that represents the dependent variable, or outcome.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Valid Types:</strong></td>
</tr>
<tr>
<td></td>
<td>• FLOAT</td>
</tr>
<tr>
<td></td>
<td>• INTEGER</td>
</tr>
<tr>
<td></td>
<td>• NUMERIC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'predictor_columns'</th>
<th>A comma-separated list of the columns in <code>input_relation</code> that represent the independent variables for the model.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Valid Types:</strong></td>
</tr>
<tr>
<td></td>
<td>• FLOAT</td>
</tr>
<tr>
<td></td>
<td>• INTEGER</td>
</tr>
<tr>
<td></td>
<td>• NUMERIC</td>
</tr>
</tbody>
</table>

If the column name contains special characters, it must use double quotes.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude_columns='col1, col2, ... coln'</td>
<td>(Optional) The columns from input_relation that you want to exclude from the input_columns argument.</td>
<td>Empty</td>
</tr>
<tr>
<td>error_tolerance=value</td>
<td>(Optional) Defines the acceptable error margin. Any data points outside this region add a penalty to the cost function.</td>
<td>0.1</td>
</tr>
<tr>
<td>C=value</td>
<td>(Optional) Sets the weight for misclassification cost. The algorithm minimizes the regularization cost and the misclassification cost.</td>
<td>1.0</td>
</tr>
<tr>
<td>epsilon=value</td>
<td>(Optional) Used to control accuracy.</td>
<td>1e-3</td>
</tr>
<tr>
<td>max_iterations=value</td>
<td>(Optional) Determines the maximum number of iterations that the algorithm performs before achieving the specified accuracy result.</td>
<td>100</td>
</tr>
</tbody>
</table>

Privileges

To use SVM_REGRESSOR, you must either be a superuser or have CREATE privileges for the schema of the output view and SELECT privileges for the input table or view. There are no privileges needed on the function itself.

See GRANT (Schema) and GRANT (Table).

Examples

This example shows how you can use the SVM_REGRESSOR function on the faithful table:
SELECT SVM_REGRESSOR('mySvmRegModel', 'faithful', 'eruptions', 'waiting' USING PARAMETERS error_tolerance=0.1, max_iterations=100);

SVM_REGRESSOR
--------------------------------------------------------------------------------
Finished in 5 iterations.
Accepted Rows: 272 Rejected Rows: 0
(1 row)

See Also

- Building an SVM for Regression Model
- SVM (Support Vector Machine) for Regression
- PREDICT_SVM_REGRESSOR
- GET_MODEL_SUMMARY
- SVM Model Attributes

UPGRADE_MODEL

Upgrades a model from a previous Vertica version. Vertica automatically runs this function during an upgrade of the database and if you run the IMPORT_MODELS function. Use this function to upgrade models during a backup or restore.

Important: Before using a machine learning function, be aware that all the ongoing transactions might be committed.

Syntax

UPGRADE_MODEL ([USING PARAMETERS [model_name='model_name']])

Parameters

'\texttt{model\_name}'  
(Optional) Identifies the model, where \texttt{model\_name} conforms to the conventions described in Identifiers. It must also be unique among all names of sequences,
tables, projections, views, and models within the same schema. Model names are case-insensitive. If no model_name is supplied, all models are upgraded.

Privileges

If you are the dbadmin, this function will upgrade all eligible models. If you are not the dbadmin, this function will upgrade all models for which you are the owner. If model_name is supplied, you must either be the dbadmin or the model owner to upgrade the model(s).

Examples

This example shows how you can use the UPGRADE_MODEL function to upgrade the myLogisticRegModel.

```sql
=> SELECT UPGRADE_MODEL(USING PARAMETERS model_name = 'myLogisticRegModel');
UPGRADE_MODEL
---------------------
1 model(s) upgrade
(1 row)
```

This example shows how you can use the UPGRADE_MODEL function to upgrade all models that a user owns.

```sql
=> SELECT UPGRADE_MODEL();
UPGRADE_MODEL
---------------------
20 model(s) upgrade
(1 row)
```
Mathematical Functions

Some of these functions are provided in multiple forms with different argument types. Except
where noted, any given form of a function returns the same data type as its argument. The
functions working with DOUBLE PRECISION data could vary in accuracy and behavior in
boundary cases depending on the host system.

ABS

Returns the absolute value of the argument. The return value has the same data type as the
argument.

Behavior Type

Immutable

Syntax

ABS ( expression )

Parameters

| expression | Is a value of type INTEGER or DOUBLE PRECISION |

Examples

```
SELECT ABS(-28.7);
-abs
------
 28.7
(1 row)
```
ACOS

Returns a DOUBLE PRECISION value representing the trigonometric inverse cosine of the argument.

Behavior Type

Immutable

Syntax

ACOS ( expression )

Parameters

| expression | Is a value of type DOUBLE PRECISION |

Example

```
SELECT ACOS (1);
acos
------
0
(1 row)
```

ASIN

Returns a DOUBLE PRECISION value representing the trigonometric inverse sine of the argument.

Behavior Type

Immutable
Syntax

\texttt{ASIN( expression )}

Parameters

| expression | Is a value of type DOUBLE PRECISION |

Example

```
SELECT ASIN(1);

asin
-------------
1.5707963267949
(1 row)
```

ATAN

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the argument.

Behavior Type

Immutable

Syntax

\texttt{ATAN( expression )}

Parameters

| expression | Is a value of type DOUBLE PRECISION |
Example

```
SELECT ATAN(1);
  atan
--------------
  0.785398163397448
(1 row)
```

ATAN2

Returns a DOUBLE PRECISION value representing the trigonometric inverse tangent of the arithmetic dividend of the arguments.

Behavior Type

Immutable

Syntax

```
ATAN2 ( quotient, divisor )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quotient</td>
<td>Is an expression of type DOUBLE PRECISION representing the quotient</td>
</tr>
<tr>
<td>divisor</td>
<td>Is an expression of type DOUBLE PRECISION representing the divisor</td>
</tr>
</tbody>
</table>

Example

```
SELECT ATAN2(2,1);
  ATAN2
----------
  1.10714871779409
(1 row)
```
CBRT

Returns the cube root of the argument. The return value has the type DOUBLE PRECISION.

Behavior Type

Immutable

Syntax

CBRT ( expression )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Value of type DOUBLE PRECISION</th>
</tr>
</thead>
</table>

Examples

```
SELECT CBRT(27.0);

 cbrt
-----
  3

(1 row)
```

CEILING (CEIL)

Rounds the returned value up to the next whole number. Any expression that contains even a slight decimal is rounded up.

Behavior Type

Immutable
Syntax

CEILING ( expression ) CEIL ( expression )

Parameters

| expression | Is a value of type INTEGER or DOUBLE PRECISION |

Notes

CEILING is the opposite of FLOOR, which rounds the returned value down:

```sql
=> SELECT CEIL(48.01) AS ceiling, FLOOR(48.01) AS floor;
+-----+-----+
<table>
<thead>
<tr>
<th>cei</th>
<th>flo</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>48</td>
</tr>
</tbody>
</table>
+-----+-----+
(1 row)
```

Examples

```sql
=> SELECT CEIL(-42.8);
      CEIL
------
     -42
(1 row)
```

COS

Returns a DOUBLE PRECISION value that represents the trigonometric cosine of the passed parameter.

Behavior Type

Immutable
Syntax

\[ \text{COS}( \text{expression} ) \]

Parameters

| expression | An expression of type DOUBLE PRECISION |

Example

```
SELECT COS(-1);
COS
------------------
 0.54030230586814  
(1 row)
```

COSH

Returns a DOUBLE PRECISION value that represents the hyperbolic cosine of the passed parameter.

Behavior Type

Immutable

Syntax

\[ \text{COSH}( \text{expression} ) \]

Parameters

| expression | An expression of type DOUBLE PRECISION |
Example

```sql
=> SELECT COSH(-1);
    COSH
---------------
1.54308063481524
```

**COT**

Returns a DOUBLE PRECISION value representing the trigonometric cotangent of the argument.

**Behavior Type**

Immutable

**Syntax**

```
COT ( expression )
```

**Parameters**

<table>
<thead>
<tr>
<th><code>expression</code></th>
<th>Is a value of type DOUBLE PRECISION</th>
</tr>
</thead>
</table>

**Example**

```sql
SELECT COT(1);
   cot
----------
0.642092615934331
(1 row)
```

**DEGREES**

Converts an expression from **RADIANS** to fractional degrees, or from degrees, minutes, and seconds to fractional degrees. The return value has the type DOUBLE PRECISION.
Behavior Type

Immutable

Syntax 1

DEGREES (radians)

Syntax 2

DEGREES (degrees, minutes, seconds)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radians</td>
<td>A unit of angular measure, $2\pi$ radians is equal to a full rotation.</td>
</tr>
<tr>
<td>degrees</td>
<td>A unit of angular measure, equal to 1/360 of a full rotation.</td>
</tr>
<tr>
<td>minutes</td>
<td>A unit of angular measurement, representing 1/60 of a degree.</td>
</tr>
<tr>
<td>seconds</td>
<td>A unit of angular measurement, representing 1/60 of a minute.</td>
</tr>
</tbody>
</table>

Examples

```sql
SELECT DEGREES(0.5);
DEGREES
---------------------
28.6478897565412
(1 row)
SELECT DEGREES(1,2,3);
DEGREES
---------------------
1.03416666666667
(1 row)
```

DISTANCE

Returns the distance (in kilometers) between two points. You specify the latitude and longitude of both the starting point and the ending point. You can also specify the radius of curvature for
greater accuracy when using an ellipsoidal model.

**Behavior Type**

Immutable

**Syntax**

\[
\text{DISTANCE} \left( \text{lat}_0, \text{lon}_0, \text{lat}_1, \text{lon}_1, \text{radius\_of\_curvature} \right)
\]

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{lat}_0 )</td>
<td>Specifies the latitude of the starting point.</td>
</tr>
<tr>
<td>( \text{lon}_0 )</td>
<td>Specifies the longitude of the starting point.</td>
</tr>
<tr>
<td>( \text{lat}_1 )</td>
<td>Specifies the latitude of the ending point</td>
</tr>
<tr>
<td>( \text{lon}_1 )</td>
<td>Specifies the longitude of the ending point</td>
</tr>
<tr>
<td>( \text{radius_of_curvature} )</td>
<td>Specifies the radius of the curvature of the earth at the midpoint between the starting and ending points. This parameter allows for greater accuracy when using an ellipsoidal earth model. If you do not specify this parameter, it defaults to the WGS-84 average r1 radius, about 6371.009 km.</td>
</tr>
</tbody>
</table>

**Example**

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is spherical.

```sql
SELECT DISTANCE(45,0,45,1);
DISTANCE
---------------------
78.6262959272162
(1 row)
```
DISTANCEV

Returns the distance (in kilometers) between two points using the Vincenty formula. Because the Vincenty formula includes the parameters of the WGS-84 ellipsoid model, you need not specify a radius of curvature. You specify the latitude and longitude of both the starting point and the ending point. This function is more accurate, but will be slower, than the DISTANCE function.

Behavior Type

Immutable

Syntax

DISTANCEV (lat0, lon0, lat1, lon1);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat0</td>
<td>Specifies the latitude of the starting point.</td>
</tr>
<tr>
<td>lon0</td>
<td>Specifies the longitude of the starting point.</td>
</tr>
<tr>
<td>lat1</td>
<td>Specifies the latitude of the ending point.</td>
</tr>
<tr>
<td>lon1</td>
<td>Specifies the longitude of the ending point.</td>
</tr>
</tbody>
</table>

Example

This example finds the distance in kilometers for 1 degree of longitude at latitude 45 degrees, assuming earth is ellipsoidal.

```
SELECT DISTANCEV(45,0, 45,1);
  distanceV
------------------------
    78.8463347095916
(1 row)
```
EXP

Returns the exponential function, \( e \) to the power of a number. The return value has the same data type as the argument.

Behavior Type

Immutable

Syntax

\( \text{EXP}(\text{exponent}) \)

Parameters

| exponent | Is an expression of type INTEGER or DOUBLE PRECISION |

Example

```
SELECT EXP(1.0);
  exp
----------
  2.71828182845905
(1 row)
```

FLOOR

Rounds the returned value down to the next whole number. For example, each of these functions evaluates to 5:

```
floor(5.01)
floor(5.5)
floor(5.99)
```
Behavior Type

Immutable

Syntax

FLOOR ( expression )

Parameters

| expression | Is an expression of type INTEGER or DOUBLE PRECISION. |

Notes

FLOOR is the opposite of CEILING, which rounds the returned value up:

```sql
=> SELECT FLOOR(48.01) AS floor, CEIL(48.01) AS ceiling;
  floor | ceiling
---------+----------
     48 |     49
      (1 row)
```

Examples

```sql
=> SELECT FLOOR((TIMESTAMP '2005-01-17 10:00' - TIMESTAMP '2005-01-01') / INTERVAL '7');
  FLOOR
---------
      2
      (1 row)
=> SELECT FLOOR(-42.8);
  FLOOR
---------
     -43
      (1 row)
=> SELECT FLOOR(42.8);
  FLOOR
---------
      42
      (1 row)
```

Although the following example looks like an INTEGER, the number on the left is $2^{49}$ as an INTEGER, but the number on the right is a FLOAT:
SELECT 1 << 49, FLOOR(1 << 49);
?column? | floor
------------------------+--------
 562949953421312 | 562949953421312
(1 row)

Compare the above example to:

SELECT 1 << 50, FLOOR(1 << 50);
?column? | floor
------------------------+--------
1125899906842624 | 1.12589990684262e+15
(1 row)

HASH

Calculates a hash value over the function arguments, producing a value in the range \( 0 \leq x < 2^{63} \).

The HASH function is typically used to segment a projection over a set of cluster nodes. The function selects a specific node for each row based on the values of the row columns. The HASH function distributes data evenly across the cluster, which facilitates optimal query execution.

Behavior Type

Immutable

Syntax

HASH ( expression-arg[,... ] )

Parameters

| expression-arg | An expression of any data type, up to 32 arguments. Any functions that are included in expression must be deterministic. For the purpose of hash segmentation, each expression typically resolves to a column reference. |
Examples

```sql
=> SELECT HASH(product_price, product_cost) FROM product_dimension
    WHERE product_price = '11';
hash
-----------------------
4157497907121511878
1799398249227328285
325022637492749639
(3 rows)
```

See Also

Hash Segmentation Clause

**LN**

Returns the natural logarithm of the argument. The return data type is the same as the argument.

**Behavior Type**

Immutable

**Syntax**

```
LN ( expression )
```

**Parameters**

<table>
<thead>
<tr>
<th>expression</th>
<th>Is an expression of type INTEGER or DOUBLE PRECISION</th>
</tr>
</thead>
</table>

**Example**

```sql
SELECT LN(2);
ln
```
LOG

Returns the logarithm to the specified base of the argument. The data type of the return value is the same data type as the passed parameter.

Behavior Type

Immutable

Syntax

\[
\text{LOG} \left( \left[ \text{base}, \right. \right. \left. \text{expression} \right) \right.
\]

Parameters

<table>
<thead>
<tr>
<th>base</th>
<th>Specifies the base (default is base 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>An expression of type INTEGER or DOUBLE PRECISION</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT LOG(2.0, 64);
LOG
-----
  6
(1 row)
SELECT LOG(100);
LOG
-----
  2
(1 row)
```
LOG10

Returns the base 10 logarithm of the argument, also known as the common logarithm. The data type of the return value is the same as the data type of the passed parameter.

Behavior Type

Immutable

Syntax

```
LOG10(expression)
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>An expression of type INTEGER or DOUBLE PRECISION</th>
</tr>
</thead>
</table>

Examples

```
=> SELECT LOG10(30);
   LOG10
-----------------
  1.47712125471966
(1 row)
```

MOD

Returns the remainder of a division operation.

Behavior Type

Immutable
Syntax

```
MOD( expression1, expression2 )
```

Parameters

<table>
<thead>
<tr>
<th><code>expression1</code></th>
<th>A numeric data type that specifies the dividend.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expression2</code></td>
<td>A numeric data type that specifies the divisor.</td>
</tr>
</tbody>
</table>

Computation Rules

When computing `MOD(expression1, expression2)`, the following rules apply:

- If either `expression1` or `expression2` is the null value, then the result is the null value.
- If `expression2` is zero, then an exception condition is raised: data exception — division by zero.
- Otherwise, the result is the unique exact numeric value `R` with scale 0 (zero) such that all of the following are true:
  - `R` has the same sign as `expression2`.
  - The absolute value of `R` is less than the absolute value of `expression1`.
  - `expression2 = expression1 * K + R` for some exact numeric value `K` with scale 0 (zero).

Examples

```
SELECT MOD(9,4);
mod
-----
1
(1 row)

SELECT MOD(10,3);
mod
-----
1
(1 row)
```
PI

Returns the constant pi (Π), the ratio of any circle's circumference to its diameter in Euclidean geometry. The return type is DOUBLE PRECISION.

Behavior Type

Immutable

Syntax

PI()

Examples

```
SELECT PI();
  pi
---------------------
  3.14159265358979
(1 row)
```
POWER (or POW)

Returns a DOUBLE PRECISION value representing one number raised to the power of another number. You can use either POWER or POW as the function name.

Behavior Type

Immutable

Syntax

POWER ( expression1, expression2 )

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>Is an expression of type DOUBLE PRECISION that represents the base</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>Is an expression of type DOUBLE PRECISION that represents the exponent</td>
</tr>
</tbody>
</table>

Example

```
SELECT POWER(9.0, 3.0);
+------------+
| power      |
+------------+
| 729        |
(1 row)
```

RADIANS

Returns a DOUBLE PRECISION value representing an angle expressed in radians. You can express the input angle in DEGREES, and optionally include minutes and seconds.

Behavior Type

Immutable
Syntax

RADIANS (degrees [, minutes, seconds])

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>degrees</td>
<td>A unit of angular measurement, representing 1/360 of a full rotation.</td>
</tr>
<tr>
<td>minutes</td>
<td>A unit of angular measurement, representing 1/60 of a degree.</td>
</tr>
<tr>
<td>seconds</td>
<td>A unit of angular measurement, representing 1/60 of a minute.</td>
</tr>
</tbody>
</table>

Examples

```sql
SELECT RADIANS(45);
RADIANS
-----------------------
0.785398163397448
(1 row)
SELECT RADIANS (1,2,3);
RADIANS
-----------------------
0.018049613347708
(1 row)
```

RANDOM

Returns a uniformly-distributed random number x, where 0 <= x < 1.

Typical pseudo-random generators accept a seed, which is set to generate a reproducible pseudo-random sequence. Vertica, however, distributes SQL processing over a cluster of nodes, where each node generates its own independent random sequence.

Results depending on RANDOM are not reproducible because the work might be divided differently across nodes. Therefore, Vertica automatically generates truly random seeds for each node each time a request is executed and does not provide a mechanism for forcing a specific seed.
Behavior Type

Volatile

Syntax

RANDOM()

Parameters

RANDOM has no arguments. Its result is a FLOAT data type (also called DOUBLE PRECISION).

Examples

In the following example, the result is a float, which is $\geq 0$ and $< 1.0$:

```
SELECT RANDOM();
random
----------------------
0.211625560652465
(1 row)
```

RANDOMINT

Accepts and returns an INTEGER value. RANDOMINT($n$) returns one of the $n$ integers from 0 through $n - 1$.

Typical pseudo-random generators accept a seed, which is set to generate a reproducible pseudo-random sequence. Vertica, however, distributes SQL processing over a cluster of nodes, where each node generates its own independent random sequence.

Results depending on RANDOM are not reproducible because the work might be divided differently across nodes. Therefore, Vertica automatically generates truly random seeds for each node each time a request is executed and does not provide a mechanism for forcing a specific seed.

Behavior Type

Volatile
Syntax

RANDOMINT ( n )

Parameters

The value accepted is any positive integer (n) between the values 1 and 9,223,372,036,854,775,807.

For general information on integer data types, refer to the section, INTEGER.

Restrictions

If you provide a negative value, or if you exceed the maximum value, Vertica returns an error.

Example

In the following example, the result is an INTEGER, which is \( \geq 0 \) and \( < n \), randomly chosen from the set \{0,1,2,3,4\}.

```sql
=> SELECT RANDOMINT(5);
RANDOMINT
----------
 3
(1 row)
```

RANDOMINT_crypto

Accepts and returns an INTEGER value. RANDOMINT_crypto (n) returns one of the n integers from 0 through n - 1. For this cryptographic random number generator, Vertica uses RAND_bytes to provide the random value.

Behavior Type

Volatile
Syntax

\texttt{RANDOMINT\_crypto ( n )}

Parameters

The value accepted is any positive integer \((n)\) between the values 1 and 9,223,372,036,854,775,807.

For general information on integer data types, see \texttt{INTEGER}.

Restrictions

If you provide a negative value, or if you exceed the maximum value, Vertica returns an error.

Examples

In the following example, notice that the result is an \texttt{INTEGER}, which is \(\geq 0\) and \(< n\), randomly chosen from the set \{0,1,2,3,4\}.

\begin{verbatim}
=> SELECT RANDOMINT\_crypto(5);
RANDOMINT\_crypto
-----------------
 3
(1 row)
\end{verbatim}

\textbf{ROUND}

Rounds a value to a specified number of decimal places, retaining the original precision and scale. Fractions greater than or equal to .5 are rounded up. Fractions less than .5 are rounded down (truncated).

Behavior Type

Immutable
Syntax

```
ROUND ( expression [, places ] )
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Is an expression of type NUMERIC or DOUBLE PRECISION (FLOAT).</th>
</tr>
</thead>
<tbody>
<tr>
<td>places</td>
<td>An INTEGER value. When places is a positive integer, Vertica rounds the value to the right of the decimal point. When places is a negative integer, Vertica rounds the value on the left side of the decimal point.</td>
</tr>
</tbody>
</table>

Notes

Using `ROUND` with a NUMERIC datatype returns NUMERIC, retaining the original precision and scale.

```
=> SELECT ROUND(3.5);
    ROUND
----------
   4.0
(1 row)
```

Examples

```
=> SELECT ROUND(2.0, 1.0) FROM dual;
    ROUND
--------
   2.0
(1 row)
=> SELECT ROUND(12.345, 2.0);
    ROUND
----------
  12.350
(1 row)
=> SELECT ROUND(3.4444444444444444);  
    ROUND
------------------
  3.0000000000000000
(1 row)
=> SELECT ROUND(3.14159, 3);  
    ROUND
------------------
3.14159
(1 row)
```
The following example creates a table with two columns, adds one row of values, and shows sample rounding to the left and right of a decimal point.

```sql
=> CREATE TABLE sampleround (roundcol1 NUMERIC, roundcol2 NUMERIC);
CREATE TABLE

=> INSERT INTO sampleround VALUES (1234567, .1234567);
OUTPUT

1
(1 row)

=> SELECT ROUND(roundcol1,-3) AS pn3, ROUND(roundcol1,-4) AS pn4, ROUND(roundcol1,-5) AS pn5 FROM sampleround;

<table>
<thead>
<tr>
<th>pn3</th>
<th>pn4</th>
<th>pn5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1235000.000000000000</td>
<td>1230000.000000000000</td>
<td>1200000.000000000000</td>
</tr>
</tbody>
</table>
(1 row)

=> SELECT ROUND(roundcol2,3) AS p3, ROUND(roundcol2,4) AS p4, ROUND(roundcol2,5) AS p5 FROM sampleround;

<table>
<thead>
<tr>
<th>p3</th>
<th>p4</th>
<th>p5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12300000000000</td>
<td>0.12350000000000</td>
<td>0.12346000000000</td>
</tr>
</tbody>
</table>
(1 row)
```

**SIGN**

Returns a DOUBLE PRECISION value of -1, 0, or 1 representing the arithmetic sign of the argument.

**Behavior Type**

Immutable
Syntax

\[ \text{SIGN (expression)} \]

Parameters

| expression | Is an expression of type DOUBLE PRECISION |

Examples

```
SELECT SIGN(-8.4);
  sign
------
   -1
(1 row)
```

SIN

Returns a DOUBLE PRECISION value that represents the trigonometric sine of the passed parameter.

Behavior Type

Immutable

Syntax

\[ \text{SIN (expression)} \]

Parameters

| expression | An expression of type DOUBLE PRECISION |
Example

```sql
SELECT SIN(30 * 2 * 3.14159 / 360);
SIN
--------------------
0.49999616987256
(1 row)
```

SINH

Returns a DOUBLE PRECISION value that represents the hyperbolic sine of the passed parameter.

Behavior Type

Immutable

Syntax

```sql
SINH ( expression )
```

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>An expression of type DOUBLE PRECISION</th>
</tr>
</thead>
</table>

Example

```sql
=> SELECT SINH(30 * 2 * 3.14159 / 360);
SINH
--------------------
0.5478529629600632
```

SQRT

Returns a DOUBLE PRECISION value representing the arithmetic square root of the argument.
Behavior Type
Immutable

Syntax
SQRT ( expression )

Parameters

| expression | Is an expression of type DOUBLE PRECISION |

Examples

```
SELECT SQRT(2);
+------------------+
<table>
<thead>
<tr>
<th>sqrt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.41421356237311</td>
</tr>
</tbody>
</table>
+------------------+
(1 row)
```

TAN

Returns a DOUBLE PRECISION value that represents the trigonometric tangent of the passed parameter.

Behavior Type
Immutable

Syntax
TAN ( expression )
Parameters

expression | An expression of type DOUBLE PRECISION

Example

=> SELECT TAN(30);
   TAN
   ------------
   -6.40533119664628
   (1 row)

TANH

Returns a DOUBLE PRECISION value that represents the hyperbolic tangent of the passed parameter.

Behavior Type

Immutable

Syntax

TANH ( expression )

Parameters

expression | An expression of type DOUBLE PRECISION

Example

=> SELECT TANH(-1);
   TANH
   ------------
TRUNC

Returns the expression value fully truncated (toward zero). Supplying a places argument truncates the expression to the number of decimal places you indicate.

Behavior Type

Immutable

Syntax

TRUNC ( expression [ , places ] )

Parameters

| expression | Is an expression of type NUMERIC or DOUBLE PRECISION (FLOAT). |
| places     | An INTEGER value. When places is a positive integer, Vertica truncates the value to the right of the decimal point. When places is a negative integer, Vertica truncates the value on the left side of the decimal point. |

Notes

Using TRUNC with a NUMERIC datatype returns NUMERIC, retaining the original precision and scale.

=> SELECT TRUNC(3.5);
  TRUNC
  -------
  3.0
  (1 row)
Examples

=> SELECT TRUNC(42.8);
TRUNC
------
  42.0
(1 row)
=> SELECT TRUNC(42.4382, 2);
TRUNC
------
  42.4300
(1 row)

The following example creates a table with two columns, adds one row of values, and shows sample truncating to the left and right of a decimal point.

=> CREATE TABLE sampletrunc (truncol1 NUMERIC, truncol2 NUMERIC);
CREATE TABLE
=> INSERT INTO sampletrunc VALUES (1234567, .1234567);
OUTPUT
-------
  1
(1 row)
=> SELECT TRUNC(truncol1,-3) AS p3, TRUNC(truncol1,-4) AS p4, TRUNC(truncol1,-5) AS p5 FROM sampletrunc;

<table>
<thead>
<tr>
<th>p3</th>
<th>p4</th>
<th>p5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234000.000000000000</td>
<td>1230000.00000000000</td>
<td>1200000.00000000000</td>
</tr>
</tbody>
</table>
(1 row)

=> SELECT TRUNC(truncol2,3) AS p3, TRUNC(truncol2,4) AS p4, TRUNC(truncol2,5) AS p5 FROM sampletrunc;

<table>
<thead>
<tr>
<th>p3</th>
<th>p4</th>
<th>p5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.123000000000000</td>
<td>0.123400000000000</td>
<td>0.123450000000000</td>
</tr>
</tbody>
</table>
(1 row)

WIDTH_BUCKET

Constructs equiwidth histograms, in which the histogram range is divided into intervals (buckets) of identical sizes. In addition, values below the low bucket return 0, and values above the high bucket return bucket_count +1. Returns an integer value.
Behavior Type

Immutable

Syntax

WIDTH_BUCKET ( expression, hist_min, hist_max, bucket_count )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>The expression for which the histogram is created. This expression must evaluate to a numeric or datetime value or to a value that can be implicitly converted to a numeric or datetime value. If expression evaluates to null, then the expression returns null.</td>
</tr>
<tr>
<td>hist_min</td>
<td>An expression that resolves to the low boundary of bucket 1. Must also evaluate to numeric or datetime values and cannot evaluate to null.</td>
</tr>
<tr>
<td>hist_max</td>
<td>An expression that resolves to the high boundary of bucket bucket_count. Must also evaluate to a numeric or datetime value and cannot evaluate to null.</td>
</tr>
<tr>
<td>bucket_count</td>
<td>An expression that resolves to a constant, indicating the number of buckets. This expression always evaluates to a positive INTEGER.</td>
</tr>
</tbody>
</table>

Notes

- WIDTH_BUCKET divides a data set into buckets of equal width. For example, Age = 0–20, 20–40, 40–60, 60–80. This is known as an equiwidth histogram.

- When using WIDTH_BUCKET pay attention to the minimum and maximum boundary values. Each bucket contains values equal to or greater than the base value of that bucket, so that age ranges of 0–20, 20–40, and so on, are actually 0–19.99 and 20–39.999.

- WIDTH_BUCKET accepts the following data types: (FLOAT and/or INTEGER), (TIMESTAMP and/or DATE and/or TIMESTAMPTZ), or (INTERVAL and/or TIME).
Examples

The following example returns five possible values and has three buckets: 0 [Up to 100), 1 [100–300), 2 [300–500), 3 [500–700), and 4 [700 and up):

```sql
SELECT product_description, product_cost, WIDTH_BUCKET(product_cost, 100, 700, 3);
```

The following example creates a nine-bucket histogram on the annual_income column for customers in Connecticut who are female doctors. The results return the bucket number to an “Income” column, divided into eleven buckets, including an underflow and an overflow. Note that if customers had an annual incomes greater than the maximum value, they would be assigned to an overflow bucket, 10:

```sql
SELECT customer_name, annual_income, WIDTH_BUCKET(annual_income, 100000, 1000000, 9) AS "Income"
FROM public.customer_dimension WHERE customer_state='CT'
AND title='Dr.' AND customer_gender='Female' AND household_id < '1000'
ORDER BY "Income";
```

In the following result set, the reason there is a bucket 0 is because buckets are numbered from 1 to bucket_count. Anything less than the given value of hist_min goes in bucket 0, and anything greater than the given value of hist_max goes in the bucket bucket_count+1. In this example, bucket 9 is empty, and there is no overflow. The value 12,283 is less than 100,000, so it goes into the underflow bucket.

<table>
<thead>
<tr>
<th>customer_name</th>
<th>annual_income</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joanna A. Nguyen</td>
<td>12283</td>
<td>0</td>
</tr>
<tr>
<td>Amy I. Nguyen</td>
<td>109806</td>
<td>1</td>
</tr>
<tr>
<td>Juanita L. Taylor</td>
<td>219002</td>
<td>2</td>
</tr>
<tr>
<td>Carla E. Brown</td>
<td>240872</td>
<td>2</td>
</tr>
<tr>
<td>Kim U. Overstreet</td>
<td>284011</td>
<td>2</td>
</tr>
<tr>
<td>Tiffany N. Reyes</td>
<td>323213</td>
<td>3</td>
</tr>
<tr>
<td>Rebecca V. Martin</td>
<td>324493</td>
<td>3</td>
</tr>
<tr>
<td>Betty . Roy</td>
<td>476055</td>
<td>4</td>
</tr>
<tr>
<td>Midori B. Young</td>
<td>462587</td>
<td>4</td>
</tr>
<tr>
<td>Martha T. Brown</td>
<td>687810</td>
<td>6</td>
</tr>
<tr>
<td>Julie D. Miller</td>
<td>616509</td>
<td>6</td>
</tr>
<tr>
<td>Julie Y. Nielson</td>
<td>894910</td>
<td>8</td>
</tr>
<tr>
<td>Sarah B. Weaver</td>
<td>896260</td>
<td>8</td>
</tr>
<tr>
<td>Jessica C. Nielson</td>
<td>861066</td>
<td>8</td>
</tr>
</tbody>
</table>

See Also

NULL-handling Functions

NULL-handling functions take arguments of any type, and their return type is based on their argument types.

COALESCE

Returns the value of the first non-null expression in the list. If all expressions evaluate to null, then COALESCE returns null.

COALESCE conforms to the ANSI SQL-92 standard.

Behavior Type

Immutable

Syntax

COALESCE ( expression[,... ] );

Example

=> SELECT product_description, COALESCE(
  lowest_competitor_price, highest_competitor_price, average_competitor_price) AS price
  FROM product_dimension LIMIT 10;

<table>
<thead>
<tr>
<th>product_description</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand #313 corn muffins</td>
<td>565</td>
</tr>
<tr>
<td>Brand #591 hot dogs</td>
<td>323</td>
</tr>
<tr>
<td>Brand #1247 american cheese</td>
<td>263</td>
</tr>
<tr>
<td>Brand #1510 whole milk</td>
<td>183</td>
</tr>
<tr>
<td>Brand #2549 vegetable soup</td>
<td>491</td>
</tr>
<tr>
<td>Brand #4520 catfish</td>
<td>113</td>
</tr>
<tr>
<td>Brand #4684 onions</td>
<td>56</td>
</tr>
<tr>
<td>Brand #6978 salmon</td>
<td>195</td>
</tr>
<tr>
<td>Brand #6924 bananas</td>
<td>80</td>
</tr>
<tr>
<td>Brand #7912 green peppers</td>
<td>180</td>
</tr>
</tbody>
</table>

(10 rows)
See Also

- CASE Expressions
- ISNULL

IFNULL

Returns the value of the first non-null expression in the list.
IFNULL is an alias of NVL.

Behavior Type

Immutable

Syntax

IFNULL ( expression1, expression2 );

Parameters

- If expression1 is null, then IFNULL returns expression2.
- If expression1 is not null, then IFNULL returns expression1.

Notes

- COALESCE is the more standard, more general function.
- IFNULL is equivalent to ISNULL.
- IFNULL is equivalent to COALESCE except that IFNULL is called with only two arguments.
- ISNULL(a, b) is different from x IS NULL.
- The arguments can have any data type supported by Vertica.
• Implementation is equivalent to the CASE expression. For example:

```sql
CASE WHEN expression1 IS NULL THEN expression2
ELSE expression1 END;
```

• The following statement returns the value 140:

```sql
SELECT IFNULL(NULL, 140) FROM employee_dimension;
```

• The following statement returns the value 60:

```sql
SELECT IFNULL(60, 90) FROM employee_dimension;
```

Examples

```sql
=> SELECT IFNULL(Score, 0.0) FROM Testing;
    IFNULL
---------
  100.0
   87.0
    .0
    .0
    .0
(5 rows)
```

See Also

• **CASE Expressions**

• **COALESCE**

• **NVL**

• **ISNULL**

### ISNULL

Returns the value of the first non-null expression in the list.

ISNULL is an alias of **NVL**.
Behavior Type

Immutable

Syntax

ISNULL ( expression1, expression2 );

Parameters

- If expression1 is null, then ISNULL returns expression2.
- If expression1 is not null, then ISNULL returns expression1.

Notes

- COALESCE is the more standard, more general function.
- ISNULL is equivalent to COALESCE except that ISNULL is called with only two arguments.
- ISNULL(a,b) is different from x IS NULL.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression. For example:

```sql
CASE WHEN expression1 IS NULL THEN expression2 ELSE expression1 END;
```

- The following statement returns the value 140:

```sql
SELECT ISNULL(NULL, 140) FROM employee_dimension;
```

- The following statement returns the value 60:

```sql
SELECT ISNULL(60, 90) FROM employee_dimension;
```
Examples

```sql
SELECT product_description, product_price, 
     ISNULL(product_cost, 0.0) AS cost 
FROM product_dimension;
```

<table>
<thead>
<tr>
<th>product_description</th>
<th>product_price</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand #59957 wheat bread</td>
<td>405</td>
<td>207</td>
</tr>
<tr>
<td>Brand #59052 blueberry muffins</td>
<td>211</td>
<td>140</td>
</tr>
<tr>
<td>Brand #59004 english muffins</td>
<td>399</td>
<td>240</td>
</tr>
<tr>
<td>Brand #53222 wheat bread</td>
<td>323</td>
<td>94</td>
</tr>
<tr>
<td>Brand #52951 croissants</td>
<td>367</td>
<td>121</td>
</tr>
<tr>
<td>Brand #58058 croissants</td>
<td>100</td>
<td>94</td>
</tr>
<tr>
<td>Brand #49398 white bread</td>
<td>318</td>
<td>25</td>
</tr>
<tr>
<td>Brand #46099 wheat bread</td>
<td>242</td>
<td>3</td>
</tr>
<tr>
<td>Brand #45283 wheat bread</td>
<td>111</td>
<td>105</td>
</tr>
<tr>
<td>Brand #43503 jelly donuts</td>
<td>259</td>
<td>19</td>
</tr>
</tbody>
</table>

(10 rows)

See Also

- [CASE Expressions](#)
- [COALESCE](#)
- [NVL](#)

### NULLIF

Compares two expressions. If the expressions are not equal, the function returns the first expression (expression1). If the expressions are equal, the function returns null.

#### Behavior Type

Immutable

#### Syntax

```sql
NULLIF( expression1, expression2 )
```
Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>Is a value of any data type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>Must have the same data type as expr1 or a type that can be implicitly cast to match expression1. The result has the same type as expression1.</td>
</tr>
</tbody>
</table>

Examples

The following series of statements illustrates one simple use of the NULLIF function.

Creates a single-column table `t` and insert some values:

```sql
CREATE TABLE t (x TIMESTAMPTZ);
INSERT INTO t VALUES('2009-09-04 09:14:00-04');
INSERT INTO t VALUES('2010-09-04 09:14:00-04');
```

Issue a select statement:

```sql
SELECT x, NULLIF(x, '2009-09-04 09:14:00 EDT') FROM t;
```

```
+-----------------+---------+
<table>
<thead>
<tr>
<th>x</th>
<th>nullif</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-09-04 09:14:00-04</td>
<td>nullif</td>
</tr>
<tr>
<td>2010-09-04 09:14:00-04</td>
<td>2010-09-04 09:14:00-04</td>
</tr>
</tbody>
</table>
```

```sql
SELECT NULLIF(1, 2);
```

```
+-----------------+
<table>
<thead>
<tr>
<th>NULLIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
```

```sql
(1 row)
```

```sql
SELECT NULLIF(20.45, 50.80);
```

```
+---------+
<table>
<thead>
<tr>
<th>NULLIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.45</td>
</tr>
</tbody>
</table>
```

(1 row)

**NULLIFZERO**

Evaluates to NULL if the value in the column is 0.
Syntax

`NULLIFZERO(expression)`

Parameters

| `expression` | (INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for 0 values. |

Example

The TESTING table below shows the test scores for 5 students. Note that test scores are missing for S. Robinson and K. Johnson (NULL values appear in the Score column.)

```
=> SELECT * FROM TESTING;
<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Doe</td>
<td>100</td>
</tr>
<tr>
<td>R. Smith</td>
<td>87</td>
</tr>
<tr>
<td>L. White</td>
<td>0</td>
</tr>
<tr>
<td>S. Robinson</td>
<td></td>
</tr>
<tr>
<td>K. Johnson</td>
<td></td>
</tr>
</tbody>
</table>
(5 rows)
```

The SELECT statement below specifies that Vertica should return any 0 values in the Score column as Null. In the results, you can see that Vertica returns L. White's 0 score as Null.

```
=> SELECT Name, NULLIFZERO(Score) FROM TESTING;
<table>
<thead>
<tr>
<th>Name</th>
<th>NULLIFZERO</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Doe</td>
<td>100</td>
</tr>
<tr>
<td>R. Smith</td>
<td>87</td>
</tr>
<tr>
<td>L. White</td>
<td></td>
</tr>
<tr>
<td>S. Robinson</td>
<td></td>
</tr>
<tr>
<td>K. Johnson</td>
<td></td>
</tr>
</tbody>
</table>
(5 rows)
```

NVL

Returns the value of the first non-null expression in the list.
Behavior Type

Immutable

Syntax

NVL ( expression1, expression2 );

Parameters

- If expression1 is null, then NVL returns expression2.
- If expression1 is not null, then NVL returns expression1.

Notes

- COALESCE is the more standard, more general function.
- NVL is equivalent to COALESCE except that NVL is called with only two arguments.
- The arguments can have any data type supported by Vertica.
- Implementation is equivalent to the CASE expression:

  CASE WHEN expression1 IS NULL THEN expression2 ELSE expression1 END;

Examples

expression1 is not null, so NVL returns expression1:

SELECT NVL('fast', 'database');
  nvl
   -----
    fast
   (1 row)

expression1 is null, so NVL returns expression2:
SELECT NVL(null, 'database');

<table>
<thead>
<tr>
<th>nvl</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
</tr>
</tbody>
</table>

(1 row)

expression2 is null, so NVL returns expression1:

SELECT NVL('fast', null);

<table>
<thead>
<tr>
<th>nvl</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
</tr>
</tbody>
</table>

(1 row)

In the following example, expression1 (title) contains nulls, so NVL returns expression2 and substitutes 'Withheld' for the unknown values:

```
SELECT customer_name, NVL(title, 'Withheld') as title
FROM customer_dimension
ORDER BY title;
```

<table>
<thead>
<tr>
<th>customer_name</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>Alexander I. Lang</td>
<td>Dr.</td>
</tr>
<tr>
<td>Steve S. Harris</td>
<td>Dr.</td>
</tr>
<tr>
<td>Daniel R. King</td>
<td>Dr.</td>
</tr>
<tr>
<td>Luigi I. Sanchez</td>
<td>Dr.</td>
</tr>
<tr>
<td>Duncan U. Carcetti</td>
<td>Dr.</td>
</tr>
<tr>
<td>Meghan K. Li</td>
<td>Dr.</td>
</tr>
<tr>
<td>Laura B. Perkins</td>
<td>Dr.</td>
</tr>
<tr>
<td>Samantha V. Robinson</td>
<td>Dr.</td>
</tr>
<tr>
<td>Joseph P. Wilson</td>
<td>Mr.</td>
</tr>
<tr>
<td>Kevin R. Miller</td>
<td>Mr.</td>
</tr>
<tr>
<td>Lauren D. Nguyen</td>
<td>Mrs.</td>
</tr>
<tr>
<td>Emily E. Goldberg</td>
<td>Mrs.</td>
</tr>
<tr>
<td>Darlene K. Harris</td>
<td>Ms.</td>
</tr>
<tr>
<td>Meghan J. Farmer</td>
<td>Ms.</td>
</tr>
<tr>
<td>Bettercare</td>
<td>Withheld</td>
</tr>
<tr>
<td>Ameristar</td>
<td>Withheld</td>
</tr>
<tr>
<td>Initech</td>
<td>Withheld</td>
</tr>
</tbody>
</table>

(17 rows)

See Also

- **CASE Expressions**
- **COALESCE**
- **ISNULL**
- **NVL2**
NVL2

Takes three arguments. If the first argument is not NULL, it returns the second argument, otherwise it returns the third argument. The data types of the second and third arguments are implicitly cast to a common type if they don't agree, similar to COALESCE.

Behavior Type

Immutable

Syntax

NVL2 ( expression1 , expression2 , expression3 );

Parameters

- If expression1 is not null, then NVL2 returns expression2.
- If expression1 is null, then NVL2 returns expression3.

Notes

Arguments two and three can have any data type supported by Vertica.

Implementation is equivalent to the CASE expression:

```
CASE WHEN expression1 IS NOT NULL THEN expression2 ELSE expression3 END;
```

Examples

In this example, expression1 is not null, so NVL2 returns expression2:

```
SELECT NVL2('very', 'fast', 'database');

+-----+
|      |
+-----+
| fast |
+-----+

(1 row)
```

In this example, expression1 is null, so NVL2 returns expression3:
SELECT NVL2(null, 'fast', 'database');
nvl2
----------
database
(1 row)

In the following example, expression1 (title) contains nulls, so NVL2 returns expression3 ('Withheld') and also substitutes the non-null values with the expression 'Known':

```
SELECT customer_name, NVL2(title, 'Known', 'Withheld')
as title
FROM customer_dimension
ORDER BY title;
```

<table>
<thead>
<tr>
<th>customer_name</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexander I. Lang</td>
<td>Known</td>
</tr>
<tr>
<td>Steve S. Harris</td>
<td>Known</td>
</tr>
<tr>
<td>Daniel R. King</td>
<td>Known</td>
</tr>
<tr>
<td>Luigi I. Sanchez</td>
<td>Known</td>
</tr>
<tr>
<td>Duncan U. Carcetti</td>
<td>Known</td>
</tr>
<tr>
<td>Meghan K. Li</td>
<td>Known</td>
</tr>
<tr>
<td>Laura B. Perkins</td>
<td>Known</td>
</tr>
<tr>
<td>Samantha V. Robinson</td>
<td>Known</td>
</tr>
<tr>
<td>Joseph P. Wilson</td>
<td>Known</td>
</tr>
<tr>
<td>Kevin R. Miller</td>
<td>Known</td>
</tr>
<tr>
<td>Lauren D. Nguyen</td>
<td>Known</td>
</tr>
<tr>
<td>Emily E. Goldberg</td>
<td>Known</td>
</tr>
<tr>
<td>Darlene K. Harris</td>
<td>Known</td>
</tr>
<tr>
<td>Meghan J. Farmer</td>
<td>Known</td>
</tr>
<tr>
<td>Bettercare</td>
<td>Withheld</td>
</tr>
<tr>
<td>Ameristar</td>
<td>Withheld</td>
</tr>
<tr>
<td>Initech</td>
<td>Withheld</td>
</tr>
</tbody>
</table>

(17 rows)

See Also

- CASE Expressions
- COALESCE
- COALESCE

ZEROIFNULL

Evaluates to 0 if the column is NULL.

Syntax

```
ZEROIFNULL(expression)
```
Parameters

| expression | (INTEGER, DOUBLE PRECISION, INTERVAL, or NUMERIC) Is the string to evaluate for NULL values. |

Example

The TESTING table below shows the test scores for 5 students. Note that L. White's score is 0, and that scores are missing for S. Robinson and K. Johnson.

```sql
=> SELECT * FROM TESTING;
Name | Score
---- | ------
J. Doe | 100
R. Smith | 87
L. White | 0
S. Robinson | 
K. Johnson | 
(5 rows)
```

The next SELECT statement specifies that Vertica should return any Null values in the Score column as 0s. In the results, you can see that Vertica returns a 0 score for S. Robinson and K. Johnson.

```sql
=> SELECT Name, ZEROIFNULL (Score) FROM TESTING;
Name | ZEROIFNULL
----- | -------
J. Doe | 100
R. Smith | 87
L. White | 0
S. Robinson | 0
K. Johnson | 0
(5 rows)
```
Pattern Matching Functions

Used with the MATCH Clause, the Vertica pattern matching functions return additional data about the patterns found/output. For example, you can use these functions to return values representing the name of the event or pattern that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- Landed on the company home page.
- Navigated to the product page.
- Ran a search.
- Clicked a link from the search results.
- Made a purchase.

For examples that use this clickstream model, see Event Series Pattern Matching in Analyzing Data.

**Note:** GROUP BY and PARTITION BY expressions do not support window functions.

See Also

- MATCH Clause
- Event Series Pattern Matching
**EVENT_NAME**

Returns a VARCHAR value representing the name of the event that matched the row.

**Syntax**

EVENT_NAME()

**Notes**

Pattern matching functions must be used in MATCH Clause syntax; for example, if you call EVENT_NAME() on its own, Vertica returns the following error message:

```sql
=> SELECT event_name();
ERROR: query with pattern matching function event_name must include a MATCH clause
```

**Example**

Note: This example uses the schema defined in Event Series Pattern Matching in Analyzing Data. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs the values for EVENT_NAME(), which is the name of the event that matched the row.

```sql
SELECT uid, sid, ts, refurl, pageurl, action, event_name()
FROM clickstream_log
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
   Entry AS refURL NOT ILIKE '%website2.com%' AND pageURL ILIKE '%website2.com%',
   Onsite AS pageURL ILIKE '%website2.com%' AND action='V',
   Purchase AS pageURL ILIKE '%website2.com%' AND action = 'P'
) PATTERN
```

Vertica Analytic Database (9.0.x)  Page 2795 of 6180
See Also

- MATCH Clause
- MATCH_ID
- PATTERN_ID
- Event Series Pattern Matching

MATCH_ID

Returns a successful pattern match as an INTEGER value. The returned value is the ordinal position of a match within a partition.

Syntax

MATCH_ID()

Notes

Pattern matching functions must be used in MATCH Clause syntax; for example, if you call MATCH_ID() on its own, Vertica returns the following error message:

```
=> SELECT match_id();
ERROR:  query with pattern matching function match_id must include a MATCH clause
```
Example

Note: This example uses the schema defined in Event Series Pattern Matching in Analyzing Data. For a more detailed example, see that topic.

The following statement analyzes users' browsing history on a site called website2.com and identifies patterns where the user reached website2.com from another Web site (Entry in the MATCH clause) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for the MATCH_ID(), which represents a sequential number of the match.

```
SELECT uid, sid, ts, refurl, pageurl, action, match_id()
FROM clickstream_log
MATCH
  (PARTITION BY uid, sid ORDER BY ts
    DEFINE
      Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
      Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
      Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
    PATTERN
      P AS (Entry Onsite* Purchase)
  )
ROWS MATCH FIRST EVENT);
```

<table>
<thead>
<tr>
<th>uid</th>
<th>sid</th>
<th>ts</th>
<th>refurl</th>
<th>pageurl</th>
<th>action</th>
<th>match_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>12:00:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:01:00</td>
<td>website2.com/home</td>
<td>website2.com/floby</td>
<td>V</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:02:00</td>
<td>website2.com/floby</td>
<td>website2.com/shamwow</td>
<td>V</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:03:00</td>
<td>website2.com/shamwow</td>
<td>website2.com/buy</td>
<td>P</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:10:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:11:00</td>
<td>website2.com/home</td>
<td>website2.com/forks</td>
<td>V</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:13:00</td>
<td>website2.com/forks</td>
<td>website2.com/buy</td>
<td>P</td>
<td>3</td>
</tr>
</tbody>
</table>

(7 rows)

See Also

- MATCH Clause
- EVENT_NAME
**PATTERN_ID**

Returns an integer value that is a partition-wide unique identifier for the instance of the pattern that matched.

**Syntax**

```
PATTERN_ID()
```

**Notes**

Pattern matching functions must be used in **MATCH Clause** syntax; for example, if call `PATTERN_ID()` on its own, Vertica returns the following error message:

```
=> SELECT pattern_id();
ERROR: query with pattern matching function pattern_id must include a MATCH clause
```

**Example**

*Note: This example uses the schema defined in Event Series Pattern Matching in Analyzing Data. For a more detailed example, see that topic.*

The following statement analyzes users' browsing history on website2.com and identifies patterns where the user landed on website2.com from another Web site (Entry) and browsed to any number of other pages (Onsite) before making a purchase (Purchase). The query also outputs values for `PATTERN_ID()`, which represents the partition-wide identifier for the instance of the pattern that matched.

```
SELECT uid, sid, ts, refurl, pageurl, action, pattern_id() FROM clickstream_log
```
MATCH
(PARTITION BY uid, sid ORDER BY ts
DEFINE
  Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
  Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
  Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
PATTERN
  P AS (Entry Onsite* Purchase)
ROWS MATCH FIRST EVENT);

<table>
<thead>
<tr>
<th>uid</th>
<th>sid</th>
<th>ts</th>
<th>refurl</th>
<th>pageurl</th>
<th>action</th>
<th>pattern_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>12:00:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:01:00</td>
<td>website2.com/home</td>
<td>website2.com/floby</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:02:00</td>
<td>website2.com/floby</td>
<td>website2.com/shamwow</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>12:03:00</td>
<td>website2.com/shamwow</td>
<td>website2.com/buy</td>
<td>P</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:10:00</td>
<td>website1.com</td>
<td>website2.com/home</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:11:00</td>
<td>website2.com/home</td>
<td>website2.com/forks</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>12:13:00</td>
<td>website2.com/forks</td>
<td>website2.com/buy</td>
<td>P</td>
<td>1</td>
</tr>
</tbody>
</table>

(7 rows)

See Also

- MATCH Clause
- EVENT_NAME
- MATCH_ID
- Event Series Pattern Matching
Regular Expression Functions

A regular expression lets you perform pattern matching on strings of characters. The regular expression syntax allows you to precisely define the pattern used to match strings, giving you much greater control than wildcard matching used in the LIKE predicate. The Vertica regular expression functions let you perform tasks such as determining if a string value matches a pattern, extracting a portion of a string that matches a pattern, or counting the number of times a pattern occurs within a string.

Vertica uses the Perl Compatible Regular Expression (PCRE) library to evaluate regular expressions. As its name implies, PCRE's regular expression syntax is compatible with the syntax used by the Perl 5 programming language. You can read PCRE's documentation about its library. However, if you are unfamiliar with using regular expressions, the Perl Regular Expressions Documentation is a good introduction.

**Note:** The regular expression functions only operate on valid UTF-8 strings. If you try using a regular expression function on a string that is not valid UTF-8, the query fails with an error. To prevent an error from occurring, use the ISUTF8 function as an initial clause to ensure the strings you pass to the regular expression functions are valid UTF-8 strings. Alternatively, or you can use the 'b' argument to treat the strings as binary octets, rather than UTF-8 encoded strings.

### ISUTF8

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as REGEXP_LIKE, which expect UTF-8 characters by default.

ISUTF8 checks for invalid UTF8 byte sequences, according to UTF-8 rules:

- invalid bytes
- an unexpected continuation byte
- a start byte not followed by enough continuation bytes
- an Overload Encoding

The presence of an invalid UTF8 byte sequence results in a return value of false.
Syntax

ISUTF8( string );

Parameters

| string | The string to test for UTF-8 compliance. |

Examples

```
=> SELECT ISUTF8(E'\xC2\xBF'); -- UTF-8 INVERTED QUESTION MARK ISUTF8
---------
t
(1 row)
=> SELECT ISUTF8(E'\xC2\xC0'); -- UNDEFINED UTF-8 CHARACTER
ISUTF8
---------
f
(1 row)
```

REGEXP_COUNT

Returns the number times a regular expression matches a string.

Syntax

REGEXP_COUNT( string, pattern [, position [, regexp_modifier ] ] )

Parameters

| string | The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a __raw__ column of a flex or columnar table, cast string to a LONG VARCHAR before searching for pattern. |
| pattern | The regular expression to search for within string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details. |
**position**

[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the $n$th character you specify.

Default value: 1

**regexp_modifier**

[Optional] One or more single-character flags that modify how the regular expression finds matches in $string$:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Treat strings as binary octets, rather than UTF-8 characters.</td>
</tr>
<tr>
<td>c</td>
<td>Force the match to be case sensitive (the default).</td>
</tr>
<tr>
<td>i</td>
<td>Force the match to be case insensitive.</td>
</tr>
<tr>
<td>m</td>
<td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line ($) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td>
</tr>
<tr>
<td>n</td>
<td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td>
</tr>
<tr>
<td>x</td>
<td>Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash () character.</td>
</tr>
</tbody>
</table>

**Notes**

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.
Examples

Count the number of occurrences of the substring *an* in the string "a man, a plan, a canal, Panama."

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an');
REGEXP_COUNT
------------
     4
(1 row)
```

Find the number of occurrences of the substring *an* in the string "a man, a plan, a canal: Panama" starting with the fifth character.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', 'an',5);
REGEXP_COUNT
------------
     3
(1 row)
```

Find the number of occurrences of a substring containing a lower-case character followed by *an*. In the first example, do not use a modifier. In the second example, use the *i* modifier to force the regular expression to ignore case.

```
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an');
REGEXP_COUNT
------------
     3
(1 row)
=> SELECT REGEXP_COUNT('a man, a plan, a canal: Panama', '[a-z]an', 1, 'i');
REGEXP_COUNT
------------
     4
```

REGEXP_ILIKE

Returns true if the string contains a match for the regular expression. This function is similar to the LIKE-predicate, except that it uses a case insensitive regular expression, rather than simple wildcard character matching.

Syntax

```
REGEXP_ILIKE( string, pattern )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <strong>raw</strong> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for pattern.</td>
</tr>
<tr>
<td>pattern</td>
<td>A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.</td>
</tr>
</tbody>
</table>

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

This example creates a table containing several strings to demonstrate regular expressions.

1. Create a table (longvc) with a single, long varchar column (body) and insert data with some distinct characters:

```sql
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE
=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödä lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べられます。それは私を傷つけません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal.');
=> insert into longvc values ('zésbaésbaa');
=> SELECT * FROM longvc;

+----------------------------------+
<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>На берегу пустынных волн</td>
</tr>
<tr>
<td>Voin syödä lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>私はガラスを食べられます。それは私を傷つけません。</td>
</tr>
<tr>
<td>Je peux manger du verre, ça ne me fait pas mal.</td>
</tr>
<tr>
<td>zésbaésbaa</td>
</tr>
</tbody>
</table>
+----------------------------------+
(5 rows)
```

2. Pattern match table rows containing a specific character ('ç'), added as part of Step 1:
3. Select all rows that contain the substring 'a':

```sql
SELECT * FROM longvc where regexp_ilike(body, 'a');
```

```
Je peux manger du verre, ça ne me fait pas mal.
```

(3 rows)

**REGEXP_INSTR**

Returns the starting or ending position in a string where a regular expression matches. This function returns 0 if no match for the regular expression is found in the string.

**Syntax**

```
REGEXP_INSTR( string, pattern [, position [, occurrence ... [, return_position [, regexp_modifier ] ... [, captured_subexp ] ] ] ] )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>string</strong></td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <strong>raw</strong> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <strong>pattern</strong>.</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <a href="https://perldoc.perl.org/Tutorial">Perl Regular Expressions Documentation</a> for details.</td>
</tr>
<tr>
<td><strong>position</strong></td>
<td>[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the <strong>n</strong>th character you specify.</td>
</tr>
<tr>
<td><strong>Default value:</strong> 1</td>
<td></td>
</tr>
<tr>
<td><strong>occurrence</strong></td>
<td>[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 1</td>
</tr>
<tr>
<td><strong>return_position</strong></td>
<td>[Optional] Sets the position within the string to return. Using the default position (0), the function returns the string position of the first character of the substring that matches the pattern. If you set return_position to 1, the function returns the position of the first character after the end of the matching substring.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 0</td>
</tr>
<tr>
<td><strong>regexp_modifier</strong></td>
<td>[Optional] One or more single-character flags that modify how the regular expression finds matches in string:</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
The captured subexpression whose position to return. By default, the function returns the position of the first character in string that matches the regular expression. If you set this value from 1 – 9, the function returns the subexpression captured by the corresponding set of parentheses in the regular expression. For example, setting this value to 3 returns the substring captured by the third set of parentheses in the regular expression.

Default value: 0

Note: The subexpressions are numbered left to right, based on the appearance of opening parenthesis, so nested regular expressions.
For example, in the regular expression \s* (\w+ \w+ (\w+)), subexpression 1 is the one that captures everything but any leading whitespaces.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

Find the first occurrence of a sequence of letters starting with the letter e and ending with the letter y in the phrase "easy come, easy go."

```sql
=> SELECT REGEXP_INSTR('easy come, easy go', 'e\w*y');
REGEXP_INSTR
-----------
    1
(1 row)
```

Find the first sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the second character (2)."

```sql
=> SELECT REGEXP_INSTR('easy come, easy go', 'e\w*y', 2);
REGEXP_INSTR
-----------
  12
(1 row)
```
Find the second sequence of letters starting with the letter e and ending with the letter y in the string "easy come, easy go" starting at the first character.

```sql
=> SELECT REGEXP_INSTR('easy come, easy go','e\w*y',1,2);
REGEXP_INSTR
---------
   12
(1 row)
```

Find the position of the first character after the first whitespace in the string "easy come, easy go."

```sql
=> SELECT REGEXP_INSTR('easy come, easy go','\s',1,1,1);
REGEXP_INSTR
---------
    6
(1 row)
```

Find the position of the start of the third word in a string by capturing each word as a subexpression, and returning the third subexpression's start position.

```sql
=> SELECT REGEXP_INSTR('one two three','(\w+)s+(\w+)s+(\w+)', 1,1,'',3);
REGEXP_INSTR
---------
    9
(1 row)
```

**REGEXP_LIKE**

Returns true if the string matches the regular expression. This function is similar to the LIKE-predicate, except that it uses regular expressions rather than simple wildcard character matching.

**Syntax**

`REGEXP_LIKE( string, pattern[, modifiers ] )`

**Parameters**

| string | The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a __raw__ column of a flex or columnar table, cast string to a LONG VARCHAR before searching for pattern. |
| pattern | A string containing the regular expression to match against the string. The |
The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>b</strong></td>
<td>Treat strings as binary octets, rather than UTF-8 characters.</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>Force the match to be case sensitive (the default).</td>
</tr>
<tr>
<td><strong>i</strong></td>
<td>Force the match to be case insensitive.</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line ($) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash () character.</td>
</tr>
</tbody>
</table>

**Notes**

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

**Examples**

This example creates a table containing several strings to demonstrate regular expressions.
CREATE TABLE t (v VARCHAR);

CREATE TABLE
CREATE PROJECTION t1 AS SELECT * FROM t;
CREATE PROJECTION
COPY t FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.

```sql
=> CREATE TABLE t (v VARCHAR);
CREATE TABLE
=> CREATE PROJECTION t1 AS SELECT * FROM t;
CREATE PROJECTION
=> COPY t FROM stdin;
```

Select all records in the table that contain the letter "a."

```sql
=> SELECT * FROM t;
  v
  ------
    aaa
    Aaa
    abc
    abc1
    123
(5 rows)
```

Select all of the rows in the table that start with the letter "a."

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v,'^a');
  v
    ------
      Aaa
      aaa
      abc
      abc1
(4 rows)
```

Select all rows that contain the substring "aa."

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v,'aa');
  v
    ------
      Aaa
      aaa
(2 rows)
```

Select all rows that contain a digit.
```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v, '\d');
 v
----
 123
 abc1
(2 rows)
```

Select all rows that contain the substring "aaa."

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa');
 v
----
 aaa
aaa
(1 row)
```

Select all rows that contain the substring "aaa" using case insensitive matching.

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'aaa', 'i');
 v
----
 Aaa
 aaa
(2 rows)
```

Select rows that contain the substring "a b c."

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c');
 v
---
(0 rows)
```

Select rows that contain the substring "a b c" ignoring space within the regular expression.

```sql
=> SELECT v FROM t WHERE REGEXP_LIKE(v, 'a b c', 'x');
 v
------
 abc
 abc1
(2 rows)
```

Add multi-line rows to demonstrate using the "m" modifier.

```sql
=> COPY t FROM stdin RECORD TERMINATOR '!';
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
>> Record 1 line 1
>> Record 1 line 2
>> Record 1 line 3!
>> Record 2 line 1
>> Record 2 line 2
>> Record 2 line 3!
>> \.r
```

Select rows that start with the substring "Record" and end with the substring "line 2."
Select rows that start with the substring "Record" and end with the substring "line 2," treating multiple lines as separate strings.

```sql
=> SELECT v from t WHERE REGEXP_LIKE(v,'^Record.*line 2$');
  v
---
(0 rows)
```

```sql
=> SELECT v from t WHERE REGEXP_LIKE(v,'^Record.*line 2$', 'm');
  v
-------------------------------
Record 2 line 1
Record 2 line 2
Record 2 line 3
Record 1 line 1
Record 1 line 2
Record 1 line 3
(2 rows)
```

**REGEXP_NOT_ILIKE**

Returns true if the string does not match the case-insensitive regular expression.

**Syntax**

```
REGEXP_NOT_ILIKE( string, pattern )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string</code></td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <code>__raw__</code> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <code>pattern</code>.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.</td>
</tr>
</tbody>
</table>

**Notes**

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.
If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

This example creates a table containing strings to demonstrate regular expressions.

1. Create a table (longvc) with a single, long varchar column (body). Then, insert data with some distinct characters, and query the table contents:

```sql
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE
=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödää lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べれます。それは私を傷つけていません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal.');
=> insert into longvc values ('zésbaésbaa');
=> SELECT * FROM longvc;

+-----------------------------------------------------+----------------+
<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>На берегу пустынных волн</td>
</tr>
<tr>
<td>Voin syödää lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>はガラスを食べれます。それは私を傷つけていません。</td>
</tr>
<tr>
<td>Je peux manger du verre, ça ne me fait pas mal.</td>
</tr>
<tr>
<td>は食べる。それは私は傷つけません。</td>
</tr>
<tr>
<td>zésbaésbaa</td>
</tr>
<tr>
<td>(5 rows)</td>
</tr>
</tbody>
</table>
```

2. Use REGEXP_NOT_ILIKE to return rows that do not contain a specific character ('ç '):

```sql
=> SELECT * FROM longvc where regexp_not_ilike(body, 'ç');

+-----------------------------------------------------+----------------+
<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voin syödää lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>zésbaésbaa</td>
</tr>
<tr>
<td>На берегу пустынных волн</td>
</tr>
<tr>
<td>はガラスを食べれます。それは私を傷つけていません。</td>
</tr>
<tr>
<td>(4 rows)</td>
</tr>
</tbody>
</table>
```

3. Pattern match all rows that do not contain the substring 'a ':

```sql
=> SELECT * FROM longvc where regexp_not_ilike(body, 'a');

+-----------------------------------------------------+----------------+
<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>На берегу пустынных волн</td>
</tr>
<tr>
<td>はガラスを食べれます。それは私を傷つけていません。</td>
</tr>
<tr>
<td>(2 rows)</td>
</tr>
</tbody>
</table>
```
**REGEXP_NOT_LIKE**

Returns true if the string does not contain a match for the regular expression. This function is a case sensitive regular expression.

**Syntax**

```
REGEXP_NOT_LIKE( string, pattern modifiers ] )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>string</strong></td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <strong>raw</strong> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <strong>pattern</strong>.</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>A string containing the regular expression to match against the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.</td>
</tr>
<tr>
<td><strong>modifiers</strong></td>
<td>[Optional] One or more single-character flags that modify how the regular expression finds matches in <strong>string</strong>:</td>
</tr>
<tr>
<td>b</td>
<td>Treat strings as binary octets, rather than UTF-8 characters.</td>
</tr>
<tr>
<td>c</td>
<td>Force the match to be case sensitive (the default).</td>
</tr>
<tr>
<td>i</td>
<td>Force the match to be case insensitive.</td>
</tr>
<tr>
<td>m</td>
<td>Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line ($) regular expression operators match line breaks (\n) within the string. Without the m modifier, the start and end of line operators match only the start and end of the string.</td>
</tr>
<tr>
<td>n</td>
<td>Allow the single character regular expression operator (.) to match a newline (\n). Without the n modifier, the . operator matches any character except a newline.</td>
</tr>
</tbody>
</table>
| x | Add comments to your regular expressions. Using the x modifier causes the function to ignore all unescaped space characters and comments in
the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

These examples demonstrate the REGEXP_NOT_LIKE regular expression function.

1. Create a table (longvc) with a single, long varchar column (body). Then, insert data with some distinct characters, and query the table contents:

```sql
=> CREATE table longvc(body long varchar (1048576));
CREATE TABLE
=> insert into longvc values ('На берегу пустынных волн');
=> insert into longvc values ('Voin syödä lasia, se ei vahingoita minua');
=> insert into longvc values ('私はガラスを食べられます。それは私を傷つけません。');
=> insert into longvc values ('Je peux manger du verre, ça ne me fait pas mal.');
=> insert into longvc values ('zésbaēsbaa');
=> SELECT * FROM longvc;

<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>На берегу пустынных волн</td>
</tr>
<tr>
<td>Voin syödä lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>私はガラスを食べられます。それは私を傷つけません。</td>
</tr>
<tr>
<td>Je peux manger du verre, ça ne me fait pas mal.</td>
</tr>
<tr>
<td>zésbaēsbaa</td>
</tr>
<tr>
<td>(5 rows)</td>
</tr>
</tbody>
</table>
```

2. Use REGEXP_NOT_LIKE to return rows that do not contain a specific character ('ç '):

```sql
=> SELECT * FROM longvc where regexp_not_like(body, 'ç');

<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voin syödä lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>zésbaēsbaa</td>
</tr>
</tbody>
</table>
```

(5 rows)
3. Return all rows that do not contain these characters ('.*ö.*ä'):

```sql
>> SELECT * FROM longvc where regexp_not_like(body, '.*ö.*ä');

<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Je peux manger du verre, ça ne me fait pas mal.</td>
</tr>
<tr>
<td>zésbaésbaa</td>
</tr>
<tr>
<td>На берегу пустых волн</td>
</tr>
<tr>
<td>Мы жарим гастроли. Это нанем мне.</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

4. Pattern match all rows that do not contain these specific characters ('z.*ešbaa'):

```sql
>> SELECT * FROM longvc where regexp_not_like(body, 'z.*ešbaa');

<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Je peux manger du verre, ça ne me fait pas mal.</td>
</tr>
<tr>
<td>Voin syödä lasia, se ei vahingoita minua</td>
</tr>
<tr>
<td>zésbaesbaa</td>
</tr>
<tr>
<td>На берегу пустых волн</td>
</tr>
<tr>
<td>Мы жарим гастроли. Это нанем мне.</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

**REGEXP_REPLACE**

Replace all occurrences of a substring that match a regular expression with another substring. It is similar to the REPLACE function, except it uses a regular expression to select the substring to be replaced.

**Syntax**

`REGEXP_REPLACE( string, target [, replacement [, position [, occurrence ... [, regexp_modifiers ] ] ] ] )`

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <strong>raw</strong> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for <code>pattern</code>.</td>
</tr>
<tr>
<td>target</td>
<td>The regular expression to search for within the string. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the <a href="https://perldoc.perl.org/perlretut.html">Perl Regular Expressions Documentation</a> for details.</td>
</tr>
</tbody>
</table>
### replacement
The string to replace matched substrings. If you do not supply a `replacement`, the function deletes matched substrings. The replacement string can contain backreferences for substrings captured by the regular expression. The first captured substring is inserted into the replacement string using \1, the second \2, and so on.

### position
[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the \textit{n}th character you specify.

**Default value:** 1

### occurrence
[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern.

**Default value:** 1

### regexp_modifier
[Optional] One or more single-character flags that modify how the regular expression finds matches in \textit{string}:

- **b**: Treat strings as binary octets, rather than UTF-8 characters.
- **c**: Force the match to be case sensitive (the default).
- **i**: Force the match to be case insensitive.
- **m**: Treat the string to match as multiple lines. Using this modifier, the start of line (^) and end of line ($) regular expression operators match line breaks (\n) within the string. Without the \textit{m} modifier, the start and end of line operators match only the start and end of the string.
- **n**: Allow the single character regular expression operator (.) to match a newline (\n). Without the \textit{n} modifier, the . operator matches any character except a newline.
- **x**: Add comments to your regular expressions. Using the \textit{x} modifier causes the function to ignore all unescaped space characters and
comments in the regular expression. Comments start with a hash (#) character and end with a newline (\n). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (\) character.

Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Another key difference between Oracle and Vertica is that Vertica can handle an unlimited number of captured subexpressions, while Oracle is limited to nine.

In Vertica, you can use \10 in the replacement pattern to access the substring captured by the tenth set of parentheses in the regular expression. In Oracle, \10 is treated as the substring captured by the first set of parentheses, followed by a zero. To force this Oracle behavior in Vertica, use the \g back reference and enclose the number of the captured subexpression in curly braces. For example, \g{1}0 is the substring captured by the first set of parentheses followed by a zero.

You can also name captured subexpressions to make your regular expressions less ambiguous. See the PCRE documentation for details.

Examples

Find groups of "word characters" (letters, numbers and underscore) ending with "thy" in the string "healthy, wealthy, and wise" and replace them with nothing.

```sql
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy');
REGEXP_REPLACE
----------
, , and wise
(1 row)
```

Find groups of word characters ending with "thy" and replace with the string "something."

```sql
=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something');
REGEXP_REPLACE
----------------
something, something, and wise
(1 row)
```
Find groups of word characters ending with "thy" and replace with the string "something" starting at the third character in the string.

=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something', 3);
REGEXP_REPLACE
-------------------------------
hesomething, something, and wise
(1 row)

Replace the second group of word characters ending with "thy" with the string "something."

=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','\w+thy', 'something', 1, 2);
REGEXP_REPLACE
-------------------------------
healthy, something, and wise
(1 row)

Find groups of word characters ending with "thy" capturing the letters before the "thy", and replace with the captured letters plus the letters "ish."

=> SELECT REGEXP_REPLACE('healthy, wealthy, and wise','(\w+)thy', 'lish');
REGEXP_REPLACE
-------------------------------
healish, wealish, and wise
(1 row)

Create a table to demonstrate replacing strings in a query.

=> CREATE TABLE customers (name varchar(50), phone varchar(11));
CREATE TABLE
=> CREATE PROJECTION customers1 AS SELECT * FROM customers;
CREATE PROJECTION
=> COPY customers FROM stdin;
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
>> Able, Adam|17815551234
>> Baker,Bob|18005551111
>> Chu,Cindy|16175559876
>> Dodd,Dinara|15083452121
>> 

Query the customers, using REGEXP_REPLACE to format the phone numbers.

=> SELECT name, REGEXP_REPLACE(phone, '^(\d)(\d{3})(\d{3})(\d{4})$','1-(\2) \3-\4') as phone FROM customers;
<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Able, Adam</td>
<td>1-(781) 555-1234</td>
</tr>
<tr>
<td>Baker, Bob</td>
<td>1-(800) 555-1111</td>
</tr>
<tr>
<td>Chu, Cindy</td>
<td>1-(617) 555-9876</td>
</tr>
<tr>
<td>Dodd, Dinara</td>
<td>1-(508) 345-2121</td>
</tr>
</tbody>
</table>
(4 rows)
REGEXP_SUBSTR

Returns the substring that matches a regular expression within a string. If no matches are found, this function returns NULL. This is different from an empty string, which the function can return if the regular expression matches a zero-length string.

Syntax

REGEXP_SUBSTR( string, pattern [, position [, occurrence [, regexp_modifier... [, captured_subexp ] ] ] ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The VARCHAR or LONG VARCHAR string to search for a regular expression pattern match. If string exists in a <strong>raw</strong> column of a flex or columnar table, cast string to a LONG VARCHAR before searching for pattern.</td>
</tr>
<tr>
<td>pattern</td>
<td>The regular expression to find a substring to extract. The syntax of the regular expression is compatible with the Perl 5 regular expression syntax. See the Perl Regular Expressions Documentation for details.</td>
</tr>
<tr>
<td>position</td>
<td>[Optional] The number of characters from the start of the string where the function should start searching for matches. By default, the function begins searching for a match at the first (leftmost) character. Setting this parameter to a value greater than 1 begins searching for a match at the nth character you specify. Default value: 1</td>
</tr>
<tr>
<td>occurrence</td>
<td>[Optional] Controls which occurrence of a pattern match in the string to return. By default, the function returns the position of the first matching substring. Use this parameter to find the position of subsequent matching substrings. For example, setting this parameter to 3 returns the position of the third substring that matches the pattern. Default value: 1</td>
</tr>
<tr>
<td>regexp_modifier</td>
<td>[Optional] One or more single-character flags that modify how the regular expression finds matches in string:</td>
</tr>
<tr>
<td>Modifier</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>b</code></td>
<td>Treat strings as binary octets, rather than UTF-8 characters.</td>
</tr>
<tr>
<td><code>c</code></td>
<td>Force the match to be case sensitive (the default).</td>
</tr>
<tr>
<td><code>i</code></td>
<td>Force the match to be case insensitive.</td>
</tr>
<tr>
<td><code>m</code></td>
<td>Treat the string to match as multiple lines. Using this modifier, the start of line (<code>^</code>) and end of line (<code>$</code>) regular expression operators match line breaks (<code>\n</code>) within the string. Without the <code>m</code> modifier, the start and end of line operators match only the start and end of the string.</td>
</tr>
<tr>
<td><code>n</code></td>
<td>Allow the single character regular expression operator (<code>.</code>) to match a newline (<code>\n</code>). Without the <code>n</code> modifier, the <code>.</code> operator matches any character except a newline.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Add comments to your regular expressions. Using the <code>x</code> modifier causes the function to ignore all unescaped space characters and comments in the regular expression. Comments start with a hash (<code>#</code>) character and end with a newline (<code>\n</code>). All spaces in the regular expression that you want to be matched in strings must be escaped with a backslash (<code>\</code>) character.</td>
</tr>
</tbody>
</table>

**captured_subexp**

[Optional] The captured subexpression whose position to return. By default, the function returns the position of the first character in `string` that matches the regular expression. If you set this value from 1 – 9, the function returns the subexpression captured by the corresponding set of parentheses in the regular expression. For example, setting this value to 3 returns the substring captured by the third set of parentheses in the regular expression.

**Default value:** 0

**Note:** The subexpressions are numbered left to right, based on the appearance of opening parenthesis, so nested regular expressions. For example, in the regular expression `\s* (\w+\s+ (\w+))`, subexpression 1 is the one that captures everything but any leading whitespaces.
Notes

This function operates on UTF-8 strings using the default locale, even if the locale has been set to something else.

If you are porting a regular expression query from an Oracle database, remember that Oracle considers a zero-length string to be equivalent to NULL, while Vertica does not.

Examples

Select the first substring of letters that end with "thy."

```sql
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy');
REGEXP_SUBSTR
-------------
healthy
(1 row)
```

Select the first substring of letters that ends with "thy" starting at the second character in the string.

```sql
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy',2);
REGEXP_SUBSTR
-------------
ealthy
(1 row)
```

Select the second substring of letters that ends with "thy."

```sql
=> SELECT REGEXP_SUBSTR('healthy, wealthy, and wise','\w+thy',1,2);
REGEXP_SUBSTR
-------------
wealthy
(1 row)
```

Return the contents of the third captured subexpression, which captures the third word in the string.

```sql
=> SELECT REGEXP_SUBSTR('one two three', '(\w+)\s+(\w+)\s+(\w+)', 1, 1, '', 3);
REGEXP_SUBSTR
-------------
three
(1 row)
```
**Sequence Functions**

The sequence functions provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

**NEXTVAL**

Returns the next value in a sequence. Call NEXTVAL after creating a sequence to initialize the sequence with its default value. Thereafter, call NEXTVAL to increment the sequence value for ascending sequences, or decrement its value for descending sequences.

**Behavior Type**

Volatile

**Syntax**

\[
\text{NEXTVAL('[[database.]schema.]sequence')}
\]

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td><code>myschema.thisDbObject</code></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>sequence</code></td>
<td>Identifies the target sequence.</td>
</tr>
</tbody>
</table>
Privileges

- SELECT privilege on sequence
- USAGE privilege on sequence schema

Restrictions

You cannot invoke NEXTVAL in a SELECT statement, in the following contexts:

- WHERE clause
- GROUP BY clause
- ORDER BY clause
- DISTINCT clause
- UNION
- Subquery

You also cannot invoke NEXTVAL to act on a sequence in:

- UPDATE or DELETE subqueries
- Views

You can use subqueries to work around some of these restrictions. For example, to use sequences with a DISTINCT clause:

```sql
=> SELECT t.col1, shift_allocation_seq.NEXTVAL FROM (SELECT DISTINCT col1 FROM av_temp1) t;
```

Examples

See Creating and Using Named Sequences in the Administrator's Guide

See Also

CURRVAL
CURRVAL

Returns the last value across all nodes that was set by `NEXTVAL` on this sequence in the current session. If `NEXTVAL` was never called on this sequence since its creation, Vertica returns an error.

**Behavior Type**

Volatile

**Syntax**

`CURRVAL([database.]schema.sequence-name)`

**Parameters**

<table>
<thead>
<tr>
<th><code>[database.]schema</code></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>myschema.myobject</code></td>
</tr>
<tr>
<td><code>sequence-name</code></td>
<td>The target sequence</td>
</tr>
</tbody>
</table>

**Privileges**

- SELECT privilege on sequence
- USAGE privilege on sequence schema

**Restrictions**

You cannot invoke CURRVAL in a SELECT statement, in the following contexts:
WHERE clause

GROUP BY clause

ORDER BY clause

DISTINCT clause

UNION

Subquery

You also cannot invoke CURRVAL to act on a sequence in:

- UPDATE or DELETE subqueries
- Views

Examples

See Creating and Using Named Sequences in the Administrator's Guide

LAST_INSERT_ID

Returns the last value of an AUTO_INCREMENT/IDENTITY column. If multiple sessions concurrently load the same table with an AUTO_INCREMENT/IDENTITY column, the function returns the last value generated for that column.

Note: This function works only with AUTO_INCREMENT/IDENTITY columns. It does not work with named sequences.

Behavior Type

Volatile

Syntax

LAST_INSERT_ID()
Privileges

- Table owner
- USAGE privileges on the table schema

Examples

See AUTO_INCREMENT and IDENTITY Sequences in the Administrator's Guide.
String Functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

This section describes functions and operators for examining and manipulating string values. Strings in this context include values of the types CHAR, VARCHAR, BINARY, and VARBINARY.

Unless otherwise noted, all of the functions listed in this section work on all four data types. As opposed to some other SQL implementations, Vertica keeps CHAR strings unpadded internally, padding them only on final output. So converting a CHAR(3) 'ab' to VARCHAR(5) results in a VARCHAR of length 2, not one with length 3 including a trailing space.

Some of the functions described here also work on data of non-string types by converting that data to a string representation first. Some functions work only on character strings, while others work only on binary strings. Many work for both. BINARY and VARBINARY functions ignore multibyte UTF-8 character boundaries.

Non-binary character string functions handle normalized multibyte UTF-8 characters, as specified by the Unicode Consortium. Unless otherwise specified, those character string functions for which it matters can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of UTF-8 characters. This is accomplished by adding "USING OCTETS" or "USING CHARACTERS" (default) as a parameter to the function.

Some character string functions are stable because in general UTF-8 case-conversion, searching and sorting can be locale dependent. Thus, LOWER is stable, while LOWERB is immutable. The USING OCTETS clause converts these functions into their "B" forms, so they become immutable. If the locale is set to collation=binary, which is the default, all string functions—except CHAR_LENGTH/CHARACTER_LENGTH, LENGTH, SUBSTR, and OVERLAY—are converted to their "B" forms and so are immutable.

BINARY implicitly converts to VARBINARY, so functions that take VARBINARY arguments work with BINARY.

ASCII

Converts the first character of a VARCHAR datatype to an INTEGER.
Behavior Type

Immutable

Syntax

ASCII ( expression )

Parameters

expression (VARCHAR) is the string to convert.

Notes

- ASCII is the opposite of the CHR function.
- ASCII operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.

Examples

This example returns employees whose last name begins with M. The ASCII equivalent of M is 77:

```sql
=> SELECT employee_last_name FROM employee_dimension
    WHERE ASCII(SUBSTR(employee_last_name, 1, 1)) = 76
    LIMIT 5;

+-----------------+
<table>
<thead>
<tr>
<th>employee_last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lewis</td>
</tr>
<tr>
<td>Lewis</td>
</tr>
<tr>
<td>Lampert</td>
</tr>
<tr>
<td>Lampert</td>
</tr>
<tr>
<td>Li</td>
</tr>
</tbody>
</table>
+-----------------+
(5 rows)
```

BIT_LENGTH

Returns the length of the string expression in bits (bytes * 8) as an INTEGER.
Behavior Type

Immutable

Syntax

BIT_LENGTH ( expression )

Parameters

| expression | (CHAR or VARCHAR or BINARY or VARBINARY) is the string to convert. |

Notes

BIT_LENGTH applies to the contents of VARCHAR and VARBINARY fields.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT BIT_LENGTH('abc'::varbinary);</td>
<td>24</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH('abc'::binary);</td>
<td>8</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(''::varbinary);</td>
<td>0</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(''::binary);</td>
<td>8</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(null::varbinary);</td>
<td></td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(null::binary);</td>
<td></td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(VARCHAR 'abc');</td>
<td>24</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(CHAR 'abc');</td>
<td>24</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(CHAR(6) 'abc');</td>
<td>48</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(VARCHAR(6) 'abc');</td>
<td>24</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(BINARY(6) 'abc');</td>
<td>48</td>
</tr>
<tr>
<td>SELECT BIT_LENGTH(BINARY 'abc');</td>
<td>24</td>
</tr>
</tbody>
</table>
SELECT BIT_LENGTH(VARBINARY 'abc'); 24
SELECT BIT_LENGTH(VARBINARY(6) 'abc'); 24

See Also

- CHARACTER_LENGTH
- LENGTH
- OCTET_LENGTH

BITCOUNT

Returns the number of one-bits (sometimes referred to as set-bits) in the given VARBINARY value. This is also referred to as the population count.

Behavior Type

Immutable

Syntax

BITCOUNT ( expression )

Parameters

| expression | (BINARY or VARBINARY) is the string to return. |

Examples

```sql
=> SELECT BITCOUNT(HEX_TO_BINARY('0x10'));
BITCOUNT
----------------
 1
(1 row)
```
=> SELECT BITCOUNT(HEX_TO_BINARY('0xF0'));
BITCOUNT
----------
  4
(1 row)
=> SELECT BITCOUNT(HEX_TO_BINARY('0xAB'));
BITCOUNT
----------
  5
(1 row)

BITSTRING_TO_BINARY

Translates the given VARCHAR bitstring representation into a VARBINARY value. This function is the inverse of TO_BITSTRING.

Behavior Type

Immutable

Syntax

BITSTRING_TO_BINARY ( expression )

Parameters

| expression | The VARCHAR string to process. |

Examples

If there are an odd number of characters in the hex value, the first character is treated as the low nibble of the first (furthest to the left) byte.

=> SELECT BITSTRING_TO_BINARY('0110000101100010');
BITSTRING_TO_BINARY
----------------------
  ab
(1 row)
BTRIM

Removes the longest string consisting only of specified characters from the start and end of a string.

Behavior Type

Immutable

Syntax

BTRIM ( expression [, characters-to-remove ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>(CHAR or VARCHAR) is the string to modify</th>
</tr>
</thead>
<tbody>
<tr>
<td>characters-to-remove</td>
<td>(CHAR or VARCHAR) specifies the characters to remove. The default is the space character.</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT BTRIM('xyxtrimyyx', 'xy');
BTRIM
-------
trim
(1 row)
```

See Also

- LTRIM
- RTRIM
- TRIM
CHARACTER_LENGTH

The CHARACTER_LENGTH() function:

- Returns the string length in UTF-8 characters for CHAR and VARCHAR columns
- Returns the string length in bytes (octets) for BINARY and VARBINARY columns
- Strips the padding from CHAR expressions but not from VARCHAR expressions
- Is identical to LENGTH() for CHAR and VARCHAR. For binary types, CHARACTER_LENGTH() is identical to OCTET_LENGTH().

Behavior Type

Immutable if USING OCTETS, stable otherwise.

Syntax

[ CHAR_LENGTH | CHARACTER_LENGTH ] ( expression ... [ USING { CHARACTERS | OCTETS } ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>(CHAR or VARCHAR) is the string to measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>USING CHARACTERS</td>
<td>Determines whether the character length is expressed in characters (the default) or octets.</td>
</tr>
<tr>
<td>OCTETS</td>
<td></td>
</tr>
</tbody>
</table>

Examples

=> SELECT CHAR_LENGTH('1234'::CHAR) USING OCTETS);
   octet_length
   -------------
   4
   (1 row)
=> SELECT CHAR_LENGTH('1234'::VARCHAR);
   char_length
   -----------
See Also

- **BIT_LENGTH**

**CHR**

Converts the first character of an INTEGER datatype to a VARCHAR.

**Behavior Type**

Immutable

**Syntax**

```sql
CHR ( expression )
```

**Parameters**

| expression | (INTEGER) is the string to convert and is masked to a single character. |

**Notes**

- **CHR** is the opposite of the **ASCII** function.
- **CHR** operates on UTF-8 characters, not only on single-byte ASCII characters. It continues to get the same results for the ASCII subset of UTF-8.
Examples

This example returns the VARCHAR datatype of the CHR expressions 65 and 97 from the employee table:

```sql
=> SELECT CHR(65), CHR(97) FROM employee;
CHR   | CHR
--------
A | a
A | a
A | a
A | a
A | a
A | a
A | a
A | a
A | a
A | a
A | a
A | a
(12 rows)
```

COLLATION

Applies a collation to two or more strings. Use COLLATION with ORDER BY, GROUP BY, and equality clauses.

Syntax

```
COLLATION ( 'expression' [, 'locale_or_collation_name' ] )
```

Parameters

<table>
<thead>
<tr>
<th>'expression'</th>
<th>Any expression that evaluates to a column name or to two or more values of type CHAR or VARCHAR.</th>
</tr>
</thead>
</table>
| 'locale_or_collation_name' | The ICU (International Components for Unicode) locale or collation name to use when collating the string. If you omit this parameter, COLLATION uses the collation associated with the session locale.  

To determine the current session locale, enter the vsql meta-command \locale:
To set the locale and collation, use \locale as follows:

```bash
=> \locale
en_US@collation=binary
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
```

**Locales**

The locale used for `COLLATION` can be one of the following:

- The default locale
- A session locale
- A locale that you specify when you call `COLLATION`. If you specify the locale, Vertica applies the collation associated with that locale to the data. `COLLATION` does not modify the collation for any other columns in the table.

For a list of valid ICU locales, go to [Locale Explorer (ICU)](URL).

**Binary and Non-Binary Collations**

The Vertica default locale is `en_US@collation=binary`, which uses *binary collation*. Binary collation compares binary representations of strings. Binary collation is fast, but it can result in a sort order where K precedes c because the binary representation of K is lower than c.

For non-binary collation, Vertica transforms the data according to the rules of the locale or the specified collation, and then applies the sorting rules. Suppose the locale collation is non-binary and you request a GROUP BY on string data. In this case, Vertica calls `COLLATION`, whether or not you specify the function in your query.

For information about collation naming, see [Collator Naming Scheme](URL).

**Examples**

**Collating GROUP BY Results**

The following examples are based on a `Premium_Customer` table that contains the following data:
SELECT * FROM Premium_Customer;

<table>
<thead>
<tr>
<th>ID</th>
<th>LName</th>
<th>FName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mc Coy</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>Mc Coy</td>
<td>Janice</td>
</tr>
<tr>
<td>3</td>
<td>McCoy</td>
<td>Jody</td>
</tr>
<tr>
<td>4</td>
<td>McCoy</td>
<td>Peter</td>
</tr>
<tr>
<td>5</td>
<td>McCoy</td>
<td>Brendan</td>
</tr>
<tr>
<td>6</td>
<td>Mccoy</td>
<td>Cameron</td>
</tr>
<tr>
<td>7</td>
<td>Mccoy</td>
<td>Lisa</td>
</tr>
</tbody>
</table>

The first statement shows how COLLATION applies the collation for the EN_US locale to the LName column for the locale EN_US. Vertica sorts the GROUP BY output as follows:

- Last names with spaces
- Last names where "coy" starts with a lowercase letter
- Last names where "Coy" starts with an uppercase letter

SELECT * FROM Premium_Customer ORDER BY COLLATION(LName, 'EN_US'), FName;

<table>
<thead>
<tr>
<th>ID</th>
<th>LName</th>
<th>FName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mc Coy</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>Mc Coy</td>
<td>Janice</td>
</tr>
<tr>
<td>6</td>
<td>Mccoy</td>
<td>Cameron</td>
</tr>
<tr>
<td>7</td>
<td>Mccoy</td>
<td>Lisa</td>
</tr>
<tr>
<td>5</td>
<td>McCoy</td>
<td>Brendan</td>
</tr>
<tr>
<td>3</td>
<td>McCoy</td>
<td>Jody</td>
</tr>
<tr>
<td>4</td>
<td>McCoy</td>
<td>Peter</td>
</tr>
</tbody>
</table>

The next statement shows how COLLATION collates the LName column for the locale LEN_AS:

- LEN indicates the language (L) is English (EN).
- AS (Alternate Shifted) instructs COLLATION that lowercase letters come before uppercase (shifted) letters.

In the results, the last names in which "coy" starts with a lowercase letter precede the last names where "Coy" starts with an uppercase letter.

SELECT * FROM Premium_Customer ORDER BY COLLATION(LName, 'LEN_AS'), FName;

<table>
<thead>
<tr>
<th>ID</th>
<th>LName</th>
<th>FName</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Mccoy</td>
<td>Cameron</td>
</tr>
<tr>
<td>7</td>
<td>Mccoy</td>
<td>Lisa</td>
</tr>
<tr>
<td>1</td>
<td>Mc Coy</td>
<td>Bob</td>
</tr>
<tr>
<td>5</td>
<td>McCoy</td>
<td>Brendan</td>
</tr>
<tr>
<td>2</td>
<td>Mc Coy</td>
<td>Janice</td>
</tr>
<tr>
<td>3</td>
<td>McCoy</td>
<td>Jody</td>
</tr>
<tr>
<td>4</td>
<td>McCoy</td>
<td>Peter</td>
</tr>
</tbody>
</table>

Comparing Strings with an Equality Clause
In the following query, COLLATION removes spaces and punctuation when comparing two strings in English. It then determines whether the two strings still have the same value after the punctuation has been removed:

```sql
=> SELECT COLLATION ('U.S.A', 'LEN_AS') = COLLATION('USA', 'LEN_AS');
```

Sorting Strings in Non-English Languages

The following table contains data that uses the German character eszett, ß:

```sql
=> SELECT * FROM t1;

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>ßstringß</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>SSstringSS</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>random1</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>random1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>random2</td>
<td>5</td>
<td>50</td>
</tr>
</tbody>
</table>
```

When you specify the collation LDE_S1:

- LDE indicates the language (L) is German (DE).
- S1 indicates the strength (S) of 1 (primary). This value indicates that the collation does not need to consider accents and case.

The query returns the data in the following order:

```sql
=> SELECT a FROM t1 ORDER BY COLLATION(a, 'LDE_S1'));

<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>random1</td>
</tr>
<tr>
<td>random1</td>
</tr>
<tr>
<td>random2</td>
</tr>
<tr>
<td>SSstringSS</td>
</tr>
<tr>
<td>ßstringß</td>
</tr>
</tbody>
</table>
```

CONCAT

Used to concatenate two strings.

Syntax

```sql
CONCAT ('string1','string2')
```
Behavior Type

Immutable

Parameters

<table>
<thead>
<tr>
<th>'string1'</th>
<th>Can be any datatype.</th>
</tr>
</thead>
<tbody>
<tr>
<td>'string2'</td>
<td></td>
</tr>
</tbody>
</table>

Restrictions

Varbinary and long varbinary types cannot be mixed with other types. These types return varbinary and long varbinary, respectively.

Similarly, long varchar types return long varchar.

Otherwise, the result is varchar.

Note: If either argument is null, concat returns null.

Example

The following simple examples use a sample table named alphabet, which contains two rows, letter1 and letter2. The contents are as follows.

```
=> CREATE TABLE alphabet (letter1 varchar(2), letter2 varchar(2));
CREATE TABLE
=> COPY alphabet FROM STDIN;
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
>> A|B
>> C|D
>> 
=> SELECT * FROM alphabet;
letter1 | letter2
---------|---------
C        | D
A        | B
(2 rows)
```

The following example concatenates the contents of the first column with a character string.
SELECT CONCAT(letter1, ' is a letter') FROM alphabet;

A is a letter
C is a letter
(2 rows)

The following example nests the CONCAT function.

SELECT CONCAT(CONCAT(letter1, ' and '), CONCAT(letter2, ' are both letters')) FROM alphabet;

C and D are both letters
A and B are both letters
(2 rows)

**DECODE**

Compares *expression* to each search value one by one. If *expression* is equal to a search, the function returns the corresponding result. If no match is found, the function returns default. If default is omitted, the function returns null.

DECODE is similar to the IF-THEN-ELSE and CASE expressions:

```sql
CASE expression
[WHEN search THEN result]
[WHEN search THEN result]
...
[ELSE default];
```

The arguments can have any data type supported by Vertica. The result types of individual results are promoted to the least common type that can be used to represent all of them. This leads to a character string type, an exact numeric type, an approximate numeric type, or a DATETIME type, where all the various result arguments must be of the same type grouping.

**Behavior Type**

Immutable

**Syntax**

DECODE ( expression, search, result [, search, result ]...[, default ] )
Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>The value to compare.</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>The value compared against expression.</td>
</tr>
<tr>
<td>result</td>
<td>The value returned, if expression is equal to search.</td>
</tr>
<tr>
<td>default</td>
<td>Optional. If no matches are found, DECODE returns default. If default is omitted, then DECODE returns NULL (if no matches are found).</td>
</tr>
</tbody>
</table>

Example

The following example converts numeric values in the weight column from the product_dimension table to descriptive values in the output.

```sql
=> SELECT product_description, DECODE(weight,
   2, 'Light',
   50, 'Medium',
   71, 'Heavy',
   99, 'Call for help',
   'N/A')
FROM product_dimension
WHERE category_description = 'Food'
AND department_description = 'Canned Goods'
AND sku_number BETWEEN 'SKU-#49750' AND 'SKU-#49999'
LIMIT 15;
```

<table>
<thead>
<tr>
<th>product_description</th>
<th>case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand #499 canned corn</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49900 fruit cocktail</td>
<td>Medium</td>
</tr>
<tr>
<td>Brand #49837 canned tomatoes</td>
<td>Heavy</td>
</tr>
<tr>
<td>Brand #49782 canned peaches</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49805 chicken noodle soup</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49944 canned chicken broth</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49819 canned chili</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49848 baked beans</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49989 minestrone soup</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49778 canned peaches</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49770 canned peaches</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49777 fruit cocktail</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49933 canned olives</td>
<td>N/A</td>
</tr>
<tr>
<td>Brand #49750 canned olives</td>
<td>Call for help</td>
</tr>
<tr>
<td>Brand #49777 canned tomatoes</td>
<td>N/A</td>
</tr>
</tbody>
</table>
```

(15 rows)
GREATEST

Returns the largest value in a list of expressions.

Behavior Type

Stable

Syntax

GREATEST( expression1, expression2, ... expression-n )

Parameters

expression1, expression2, and expression-n are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples.
- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

Examples

This example returns 9 as the greatest in the list of expressions:

=> SELECT GREATEST(7, 5, 9);
   GREATEST
   -------
   9
   (1 row)

Note that putting quotes around the integer expressions returns the same result as the first example:
=> SELECT GREATEST('7', '5', '9');
GREATEST
--------
 9
(1 row)

The next example returns FLOAT 1.5 as the greatest because the integer is implicitly cast to float:

=> SELECT GREATEST(1, 1.5);
GREATEST
--------
 1.5
(1 row)

The following example returns 'vertica' as the greatest:

=> SELECT GREATEST('vertica', 'analytic', 'database');
GREATEST
-------
vertica
(1 row)

Notice this next command returns NULL:

=> SELECT GREATEST('vertica', 'analytic', 'database', null);
GREATEST
--------

(1 row)

And one more:

=> SELECT GREATEST('sit', 'site', 'sight');
GREATEST
--------
site
(1 row)

See Also

- LEAST

GREATESTB

Returns its greatest argument, using binary ordering, not UTF-8 character ordering.
Behavior Type

Immutable

Syntax

GREATESTB(expression1, expression2, ... expression-n)

Parameters

expression1, expression2, and expression-n are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples.
- A NULL value in any one of the expressions returns NULL.
- Depends on the collation setting of the locale.

Examples

The following command selects straße as the greatest in the series of inputs:

```sql
=> SELECT GREATESTB('straße', 'strasse');
GREATESTB
---------
straße
(1 row)
```

This example returns 9 as the greatest in the list of expressions:

```sql
=> SELECT GREATESTB(7, 5, 9);
GREATESTB
---------
9
(1 row)
```

Note that putting quotes around the integer expressions returns the same result as the first example:
The next example returns FLOAT 1.5 as the greatest because the integer is implicitly cast to float:

```
=> SELECT GREATESTB(1, 1.5);
GREATESTB
----------
1.5
(1 row)
```

The following example returns `vertica` as the greatest:

```
=> SELECT GREATESTB('vertica', 'analytic', 'database');
GREATESTB
----------
vertica
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT GREATESTB('vertica', 'analytic', 'database', NULL);
GREATESTB
----------
(1 row)
```

And one more:

```
=> SELECT GREATESTB('sit', 'site', 'sight');
GREATESTB
----------
site
(1 row)
```

See Also

- LEASTB

HEX_TO_BINARY

Translates the given VARCHAR hexadecimal representation into a VARBINARY value.
Behavior Type

Immutable

Syntax

HEX_TO_BINARY ( [ 0x ] expression )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>(BINARY or VARBINARY) String to translate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>Optional prefix.</td>
</tr>
</tbody>
</table>

Notes

VARBINARY HEX_TO_BINARY(VARCHAR) converts data from character type in hexadecimal format to binary type. This function is the inverse of TO_HEX.

<table>
<thead>
<tr>
<th>HEX_TO_BINARY(TO_HEX(x)) = x</th>
<th>TO_HEX(HEX_TO_BINARY(x)) = x</th>
</tr>
</thead>
</table>

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low nibble of the first (furthest to the left) byte.

Examples

If the given string begins with "0x" the prefix is ignored. For example:

```sql
=> SELECT HEX_TO_BINARY('0x6162') AS hex1, HEX_TO_BINARY('6162') AS hex2;
   hex1  | hex2
-------|-------
   ab    | ab    
(1 row)
```

If an invalid hex value is given, Vertica returns an “invalid binary representation” error; for example:
SELECT HEX_TO_BINARY('0xffgf');
ERROR: invalid hex string "0xffgf"

See Also

- TO_HEX

HEX_TO_INTEGER

Translates the given VARCHAR hexadecimal representation into an INTEGER value. Vertica completes this conversion as follows:

- Adds the 0x prefix if it is not specified in the input
- Casts the VARCHAR string to a NUMERIC
- Casts the NUMERIC to an INTEGER

Behavior Type

Immutable

Syntax

HEX_TOINTEGER ([ 0x ] expression)

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>VARCHAR is the string to translate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>Is the optional prefix.</td>
</tr>
</tbody>
</table>

Examples

You can enter the string with or without the Ox prefix. For example:
If you pass the function an invalid hex value, Vertica returns an invalid input syntax error; for example:

```
=> SELECT HEX_TO_INTEGER ('0xffgf');
ERROR 3691: Invalid input syntax for numeric: "0xffgf"
```

See Also

- `TO_HEX`
- `HEX_TO_BINARY`

**INET_ATON**

Returns an integer that represents the value of the address in host byte order, given the dotted-quad representation of a network address as a string.

**Behavior Type**

Immutable

**Syntax**

```
INET_ATON ( expression )
```

**Parameters**

<table>
<thead>
<tr>
<th>expression</th>
<th>(VARCHAR) is the string to convert.</th>
</tr>
</thead>
</table>

Notes

The following syntax converts an IPv4 address represented as the string A to an integer I. INET_ATON trims any spaces from the right of A, calls the Linux function inet_pton, and converts the result from network byte order to host byte order using htonl.

```sql
=> INET_ATON(VARCHAR A) -> INT8 I
```

If A is NULL, too long, or inet_pton returns an error, the result is NULL.

Examples

The generated number is always in host byte order. In the following example, the number is calculated as 209×256^3 + 207×256^2 + 224×256 + 40.

```sql
=> SELECT INET_ATON('209.207.224.40');
    inet_aton
    -----------
     3520061480
    (1 row)

=> SELECT INET_ATON('1.2.3.4');
    inet_aton
    -----------
      16909060
    (1 row)

=> SELECT TO HEX(INET_ATON('1.2.3.4'));
    to_hex
    -------
      1020304
    (1 row)
```

See Also

- INET_NTOA

INET_NTOA

Returns the dotted-quad representation of the address as a VARCHAR, given a network address as an integer in network byte order.
Behavior Type

Immutable

Syntax

INET_NTOA ( expression )

Parameters

| expression | (INTEGER) is the network address to convert. |

Notes

The following syntax converts an IPv4 address represented as integer I to a string A.

INET_NTOA converts I from host byte order to network byte order using htonl, and calls the Linux function inet_ntop.

=> INET_NTOA(INT8 I) -> VARCHAR A

If I is NULL, greater than 2^32 or negative, the result is NULL.

Examples

=> SELECT INET_NTOA(16909060);
  inet_ntoa
  -----------
  1.2.3.4
  (1 row)

=> SELECT INET_NTOA(03021962);
  inet_ntoa
  -----------
  0.46.28.138
  (1 row)
See Also

- INET_ATON

INITCAP

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase.

Behavior Type

Immutable

Syntax

INITCAP (expression)

Parameters

| expression | (VARCHAR) is the string to format. |

Notes

- Depends on collation setting of the locale.
- INITCAP is restricted to 32750 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT INITCAP('high speed database');</td>
<td>High Speed Database</td>
</tr>
<tr>
<td>SELECT INITCAP('LINUX TUTORIAL');</td>
<td>Linux Tutorial</td>
</tr>
</tbody>
</table>
INITCAP

Capitalizes first letter of each alphanumeric word and puts the rest in lowercase. Multibyte characters are not converted and are skipped.

Behavior Type

Immutable

Syntax

INITCAP ( expression )

Parameters

expression (VARCHAR) is the string to format.

Notes

Depends on collation setting of the locale.

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT INITCAP('étudiant');</td>
<td>étudiant</td>
</tr>
<tr>
<td>SELECT INITCAP('high speed database');</td>
<td>High Speed Database</td>
</tr>
<tr>
<td>SELECT INITCAP('LINUX TUTORIAL');</td>
<td>Linux Tutorial</td>
</tr>
<tr>
<td>SELECT INITCAP('abc DEF 123aVC 124Btd,1AsT');</td>
<td>Abc Def 123Avc 124Btd,Last</td>
</tr>
</tbody>
</table>
**INSERT**

Inserts a character string into a specified location in another character string.

**Syntax**

\[
\text{INSERT( 'string1', n, m, 'string2' )}
\]

**Behavior Type**

Immutable

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{string1}</td>
<td>(VARCHAR) Is the string in which to insert the new string.</td>
</tr>
<tr>
<td>\textit{n}</td>
<td>A character of type INTEGER that represents the starting point for the insertion within \textit{string1}. You specify the number of characters from the first character in \textit{string1} as the starting point for the insertion. For example, to insert characters before &quot;c&quot;, in the string &quot;abcdef,&quot; enter 3.</td>
</tr>
<tr>
<td>\textit{m}</td>
<td>A character of type INTEGER that represents the number of characters in \textit{string1} (if any) that should be replaced by the insertion. For example, if you want the insertion to replace the letters &quot;cd&quot; in the string &quot;abcdef,&quot; enter 2.</td>
</tr>
<tr>
<td>\textit{string2}</td>
<td>(VARCHAR) Is the string to be inserted.</td>
</tr>
</tbody>
</table>

**Example**

The following example changes the string Warehouse to Storehouse using the INSERT function:

\[
\text{=> SELECT INSERT ('Warehouse',1,3,'Stor');}
\]

\[
\text{INSERT --------}
\]
INSTR

Searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence. The return value is based on the character position of the identified character.

Behavior Type

Immutable

Syntax

INSTR ( string , substring [, position [, occurrence ] ] )

Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>(CHAR or VARCHAR, or BINARY or VARBINARY) Text expression to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>(CHAR or VARCHAR, or BINARY or VARBINARY) String to search for.</td>
</tr>
<tr>
<td>position</td>
<td>Nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first character of string occupies the default position 1, and position cannot be 0.</td>
</tr>
<tr>
<td>occurrence</td>
<td>Integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.</td>
</tr>
</tbody>
</table>

Notes

Both position and occurrence must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first character of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in characters.
If the search is unsuccessful (that is, if substring does not appear occurrence times after the position character of string, the return value is 0.

Examples

The first example searches forward in string ‘abc’ for substring ‘b’. The search returns the position in ‘abc’ where ‘b’ occurs, or position 2. Because no position parameters are given, the default search starts at ‘a’, position 1.

```sql
=> SELECT INSTR('abc', 'b');
INSTR
-------
 2
(1 row)
```

The following three examples use character position to search backward to find the position of a substring.

Note: Although it might seem intuitive that the function returns a negative integer, the position of n occurrence is read left to right in the string, even though the search happens in reverse (from the end—or right side—of the string).

In the first example, the function counts backward one character from the end of the string, starting with character ‘c’. The function then searches backward for the first occurrence of ‘a’, which it finds it in the first position in the search string.

```sql
=> SELECT INSTR('abc', 'a', -1);
INSTR
-------
 1
(1 row)
```

In the second example, the function counts backward one byte from the end of the string, starting with character ‘c’. The function then searches backward for the first occurrence of ‘a’, which it finds it in the first position in the search string.

```sql
=> SELECT INSTR(VARBINARY 'abc', VARBINARY 'a', -1);
INSTR
-------
 1
(1 row)
```

In the third example, the function counts backward one character from the end of the string, starting with character ‘b’, and searches backward for substring ‘bc’, which it finds in the second position of the search string.
In the fourth example, the function counts backward one character from the end of the string, starting with character ‘b’, and searches backward for substring ‘bcef’, which it does not find. The result is 0.

```
=> SELECT INSTR('abcb', 'bc', -1);
  INSTR
    -----  
       2 
(1 row)
```

In the fifth example, the function counts backward one byte from the end of the string, starting with character ‘b’, and searches backward for substring ‘bcef’, which it does not find. The result is 0.

```
=> SELECT INSTR('abcb', 'bcef', -1);
  INSTR
    ----- 
      0 
(1 row)
```

Multibyte characters are treated as a single character:

```
=> SELECT INSTR(VARBINARY 'abcb', VARBINARY 'bcef', -1);
  INSTR
    ----- 
      0 
(1 row)
```

Use INSTRB to treat multibyte characters as binary:

```
=> SELECT INSTRB('aébc', 'b');
  INSTRB
    ----- 
      3 
(1 row)
```

**INSTRB**

Searches string for substring and returns an integer indicating the octet position within string that is the first occurrence. The return value is based on the octet position of the identified byte.
Behavior Type
Immutable

Syntax
INSTRB ( string , substring [, position [, occurrence ] ] )

Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>Is the text expression to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>Is the string to search for.</td>
</tr>
<tr>
<td>position</td>
<td>Is a nonzero integer indicating the character of string where Vertica begins the search. If position is negative, then Vertica counts backward from the end of string and then searches backward from the resulting position. The first byte of string occupies the default position 1, and position cannot be 0.</td>
</tr>
<tr>
<td>occurrence</td>
<td>Is an integer indicating which occurrence of string Vertica searches. The value of occurrence must be positive (greater than 0), and the default is 1.</td>
</tr>
</tbody>
</table>

Notes
Both position and occurrence must be of types that can resolve to an integer. The default values of both parameters are 1, meaning Vertica begins searching at the first byte of string for the first occurrence of substring. The return value is relative to the beginning of string, regardless of the value of position, and is expressed in octets.

If the search is unsuccessful (that is, if substring does not appear occurrence times after the position character of string, then the return value is 0.

Example
```sql
=> SELECT INSTRB('straße', 'ß');
INSTRB
-------
5
```
See Also

- INSTR

**ISUTF8**

Tests whether a string is a valid UTF-8 string. Returns true if the string conforms to UTF-8 standards, and false otherwise. This function is useful to test strings for UTF-8 compliance before passing them to one of the regular expression functions, such as `REGEXP_LIKE`, which expect UTF-8 characters by default.

ISUTF8 checks for invalid UTF8 byte sequences, according to UTF-8 rules:

- invalid bytes
- an unexpected continuation byte
- a start byte not followed by enough continuation bytes
- an Overload Encoding

The presence of an invalid UTF8 byte sequence results in a return value of false.

**Syntax**

`ISUTF8( string );`

**Parameters**

| `string` | The string to test for UTF-8 compliance. |

**Examples**

```sql
=> SELECT ISUTF8(E'\xC2\xBF'); -- UTF-8 INVERTED QUESTION MARK ISUTF8
--------
```
LEAST

Returns the smallest value in a list of expressions.

Behavior Type

Stable

Syntax

LEAST ( expression1, expression2, ... expression-n )

Parameters

expression1, expression2, and expression-n are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

Examples

This example returns 5 as the least:

```sql
=> SELECT LEAST(7, 5, 9);
LEAST
-----
 5
(1 row)
```
Putting quotes around the integer expressions returns the same result as the first example:

```
=> SELECT LEAST('7', '5', '9');
 LEAST
  ------
   5
(1 row)
```

In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```
=> SELECT LEAST(2, 1.5);
 LEAST
  ------
   1.5
(1 row)
```

The following example returns 'analytic' as the least:

```
=> SELECT LEAST('vertica', 'analytic', 'database');
 LEAST
   -------
   analytic
(1 row)
```

Notice this next command returns NULL:

```
=> SELECT LEAST('vertica', 'analytic', 'database', null);
 LEAST
   ------
(1 row)
```

And one more:

```
=> SELECT LEAST('sit', 'site', 'sight');
 LEAST
   ------
   sight
(1 row)
```

See Also

- **GREATEST**

**LEASTB**

Returns the function's least argument, using binary ordering, not UTF-8 character ordering.
Behavior Type

Immutable

Syntax

LEASTB ( expression1, expression2, ... expression-n )

Parameters

expression1, expression2, and expression-n are the expressions to be evaluated.

Notes

- Works for all data types, and implicitly casts similar types. See Examples below.
- A NULL value in any one of the expressions returns NULL.

Examples

The following command selects strasse as the least in the series of inputs:

```sql
=> SELECT LEASTB('straße', 'strasse');
LEASTB
---------
strasse
(1 row)
```

This example returns 5 as the least:

```sql
==> SELECT LEASTB(7, 5, 9);
LEASTB
-------
5
(1 row)
```

Putting quotes around the integer expressions returns the same result as the first example:

```sql
=> SELECT LEASTB('7', '5', '9');
LEASTB
-------
5
```
In the above example, the values are being compared as strings, so '10' would be less than '2'.

The next example returns 1.5, as INTEGER 2 is implicitly cast to FLOAT:

```sql
=> SELECT LEASTB(2, 1.5);
      leastb
  -------
     1.5
(1 row)
```

The following example returns 'analytic' as the least in the series of inputs:

```sql
=> SELECT LEASTB('vertica', 'analytic', 'database');
        leastb
  ---------
       analytic
(1 row)
```

Notice this next command returns NULL:

```sql
=> SELECT LEASTB('vertica', 'analytic', 'database', null);
      leastb
  -------
    (null)
(1 row)
```

See Also

- GREATESTB

LEFT

Returns the specified characters from the left side of a string.

Behavior Type

Immutable

Syntax

`LEFT ( string-expr, length )`
Parameters

<table>
<thead>
<tr>
<th>string-expr</th>
<th>The string expression to return.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>An integer value that specifies how many characters to return.</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT LEFT('vertica', 3);

LEFT
-------
ver
(1 row)
```

```sql
SELECT DISTINCT(
  LEFT (customer_name, 4)) FnameTruncated
FROM customer_dimension ORDER BY FnameTruncated LIMIT 10;

FnameTruncated
--------------
Alex
Amer
Amy
Anna
Barb
Ben
Bett
Bria
Carl
Crai
(10 rows)
```

See Also

SUBSTR

LENGTH

Returns the length of a string. The behavior of LENGTH varies according to the input data type:

- CHAR and VARCHAR: Identical to CHARACTER_LENGTH, returns the string length in UTF-8 characters.
- CHAR: Strips padding.
- **BINARY and VARBINARY**: Identical to **OCTET_LENGTH**, returns the string length in bytes (octets).

### Behavior Type

**Immutable**

### Syntax

```
LENGTH ( expression )
```

### Parameters

| expression | String to evaluate, one of the following: CHAR, VARCHAR, BINARY or VARBINARY. |

### Examples

<table>
<thead>
<tr>
<th>Statement</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT LENGTH('1234'::CHAR(10));</td>
<td>4</td>
</tr>
<tr>
<td>SELECT LENGTH('1234'::VARCHAR(10));</td>
<td>6</td>
</tr>
<tr>
<td>SELECT LENGTH('1234'::BINARY(10));</td>
<td>10</td>
</tr>
<tr>
<td>SELECT LENGTH('1234'::VARBINARY(10));</td>
<td>6</td>
</tr>
<tr>
<td>SELECT LENGTH(NULL::CHAR(10)) IS NULL;</td>
<td>t</td>
</tr>
</tbody>
</table>

### See Also

**BIT_LENGTH**

**LOWER**

Returns a VARCHAR value containing the argument converted to lowercase letters.
LOWER treats the string argument as a UTF-8 encoded string, rather than depending on the collation setting of the locale (for example, collation=binary) to identify the encoding.

**Behavior Type**

stable

**Syntax**

LOWER ( expression )

**Parameters**

| expression | CHAR or VARCHAR string to convert |

**Notes**

LOWER is restricted to 32500 octet inputs, since it is possible for the UTF-8 representation of result to double in size.

**Examples**

```sql
=> SELECT LOWER('AbCdEfG');
   LOWER
  -------
     abcdefg
    (1 row)
=> SELECT LOWER('The Bat In The Hat');
   LOWER
  ---------------
     the bat in the hat
    (1 row)
=> SELECT LOWER('ÉTUDIANT');
   LOWER
  -------
     étudiant
    (1 row)
```
LOWERB

Returns a character string with each ASCII character converted to lowercase. Multi-byte characters are skipped and not converted.

Behavior Type

Immutable

Syntax

LOWERB ( expression )

Parameters

| expression | CHAR or VARCHAR string to convert |

Examples

In the following example, the multi-byte UTF-8 character É is not converted to lowercase:

```sql
=> SELECT LOWERB('ÉTUDIANT');
  LOWERB
  -------
  étudiant
  (1 row)
=> SELECT LOWERB('ÉTUDIAN');
  LOWERB
  -------
  étudiant
  (1 row)
=> SELECT LOWERB('AbCdEfG');
  LOWERB
  -------
  abcdefg
  (1 row)
=> SELECT LOWERB('The Vertica Database');
  LOWERB
  ------------------
  the vertica database
  (1 row)
```
LPAD

Returns a VARCHAR value representing a string of a specific length filled on the left with specific characters.

Behavior Type

Immutable

Syntax

LPAD ( expression, length [ , fill ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>(CHAR OR VARCHAR) specifies the string to fill</td>
</tr>
<tr>
<td>length</td>
<td>(INTEGER) specifies the number of characters to return</td>
</tr>
<tr>
<td>fill</td>
<td>(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.</td>
</tr>
</tbody>
</table>

Examples

=> SELECT LPAD('database', 15, 'xzy');

LPAD

```
xzyxzyxdatabase
```

(1 row)

If the string is already longer than the specified length it is truncated on the right:

=> SELECT LPAD('establishment', 10, 'abc');

LPAD

```
establishm
```

(1 row)
LTRIM

Returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).

Behavior Type

Immutable

Syntax

LTRIM ( expression [ , characters ] )

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>expression</strong></td>
<td>(CHAR or VARCHAR) is the string to trim</td>
</tr>
<tr>
<td><strong>characters</strong></td>
<td>(CHAR or VARCHAR) specifies the characters to remove from the left side of expression. The default is the space character.</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT LTRIM('zzyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy...
**MD5**

Calculates the MD5 hash of string, returning the result as a VARCHAR string in hexadecimal.

**Behavior Type**

Immutable

**Syntax**

`MD5 ( string )`

**Parameters**

| string | Is the argument string. |

**Examples**

```
=> SELECT MD5('123');
   ----------------------------------------
   MD5
   202cb962ac59075b964b07152d234b70
(1 row)
=> SELECT MD5('Vertica':bytea);
   ----------------------------------------
   MD5
   fc45b815747d8236f9f6f6f6b9c2c3f676
(1 row)
```

**See Also**

- SHA1
- SHA224
- SHA256
- SHA384
- SHA512

**OCTET_LENGTH**

Takes one argument as an input and returns the string length in octets for all string types.

**Behavior Type**

Immutable

**Syntax**

OCTET_LENGTH ( expression )

**Parameters**

| expression | (CHAR or VARCHAR or BINARY or VARBINARY) is the string to measure. |

**Notes**

- If the data type of expression is a CHAR, VARCHAR or VARBINARY, the result is the same as the actual length of expression in octets. For CHAR, the length does not include any trailing spaces.

- If the data type of expression is BINARY, the result is the same as the fixed-length of expression.

- If the value of expression is NULL, the result is NULL.

**Examples**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
</table>
See Also

- BIT_LENGTH
- CHARACTER_LENGTH
- LENGTH

OVERLAY

Returns a VARCHAR value representing a string having had a substring replaced by another string.

Behavior Type

Immutable if using OCTETS, Stable otherwise
Syntax

OVERLAY ( expression1 PLACING expression2 FROM position
... [ FOR extent ]
... [ USING { CHARACTERS | OCTETS } ] )

Parameters

<table>
<thead>
<tr>
<th>expression1</th>
<th>(CHAR or VARCHAR) is the string to process</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression2</td>
<td>(CHAR or VARCHAR) is the substring to overlay</td>
</tr>
<tr>
<td>position</td>
<td>(INTEGER) is the character or octet position (counting from one) at which to begin the overlay</td>
</tr>
<tr>
<td>extent</td>
<td>(INTEGER) specifies the number of characters or octets to replace with the overlay</td>
</tr>
<tr>
<td>USING CHARACTERS</td>
<td>OCTETS</td>
</tr>
</tbody>
</table>

Examples

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2);
  overlay
  1xxx56789
  (1 row)

=> SELECT OVERLAY('123456789' PLACING 'XXX' FROM 2 USING OCTETS);
  overlayb
  1XXX56789
  (1 row)

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 4);
  overlay
  1xxx6789
  (1 row)

=> SELECT OVERLAY('123456789' PLACING 'xxx' FROM 2 FOR 5);
  overlay
  1xxx789
  (1 row)
OVERLAY

Returns an octet value representing a string having had a substring replaced by another string.

Behavior Type

Immutable

Syntax

OVERLAYB ( expression1, expression2, position [ , extent ] )

Parameters

| expression1 | (CHAR or VARCHAR) is the string to process |
| expression2 | (CHAR or VARCHAR) is the substring to overlay |
| position    | (INTEGER) is the octet position (counting from one) at which to begin the overlay |
| extent      | (INTEGER) specifies the number of octets to replace with the overlay |

Notes

The OVERLAYB function treats the multibyte character string as a string of octets (bytes) and use octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they treated as if each byte was a separate character.
Examples

=> SELECT OVERLAYB('123456789', 'ééé', 2);
  OVERLAYB
  ----------
  1éé89
(1 row)
=> SELECT OVERLAYB('123456789', 'ßßß', 2);
  OVERLAYB
  ----------
  1ßß89
(1 row)
=> SELECT OVERLAYB('123456789', 'xxx', 2);
  OVERLAYB
  ----------
  1xxx56789
(1 row)
=> SELECT OVERLAYB('123456789', 'xxx', 2, 4);
  OVERLAYB
  ----------
  1xxx6789
(1 row)
=> SELECT OVERLAYB('123456789', 'xxx', 2, 5);
  OVERLAYB
  ----------
  1xxx789
(1 row)
=> SELECT OVERLAYB('123456789', 'xxx', 2, 6);
  OVERLAYB
  ----------
  1xxx89
(1 row)

POSITION

Returns an INTEGER value representing the character location of a specified substring with a string (counting from one).

Behavior Type

Immutable

Syntax 1

POSITION ( substring IN string [ USING { CHARACTERS | OCTETS } ] )
### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>substring</code></td>
<td>(CHAR or VARCHAR) is the substring to locate</td>
</tr>
<tr>
<td><code>string</code></td>
<td>(CHAR or VARCHAR) is the string in which to locate the substring</td>
</tr>
<tr>
<td>`USING CHARACTERS</td>
<td>OCTETS`</td>
</tr>
</tbody>
</table>

### Syntax 2

```sql
POSITION ( substring IN string )
```

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>substring</code></td>
<td>(VARBINARY) is the substring to locate</td>
</tr>
<tr>
<td><code>string</code></td>
<td>(VARBINARY) is the string in which to locate the substring</td>
</tr>
</tbody>
</table>

### Notes

- When the string and substring are CHAR or VARCHAR, the return value is based on either the character or octet position of the substring.
- When the string and substring are VARBINARY, the return value is always based on the octet position of the substring.
- The string and substring must be consistent. Do not mix VARBINARY with CHAR or VARCHAR.
- POSITION is similar to `STRPOS` although POSITION allows finding by characters and by octet.
- If the string is not found, the return value is zero.
Examples

```sql
=> SELECT POSITION('é' IN 'étudiant' USING CHARACTERS);
  position
  --------
   1
(1 row)
=> SELECT POSITION('ß' IN 'straße' USING OCTETS);
  position
  --------
   5
(1 row)
=> SELECT POSITION('c' IN 'abcd' USING CHARACTERS);
  position
  --------
   3
(1 row)
=> SELECT POSITION(VARBINARY '456' IN VARBINARY '123456789');
  position
  --------
   4
(1 row)
```

```sql
SELECT POSITION('n' in 'León') as 'default',
  POSITIONB('León', 'n') as 'POSITIONB',
  POSITION('n' in 'León' USING CHARACTERS) as 'pos_chars',
  POSITION('n' in 'León' USING OCTETS) as 'pos_oct',INSTR('León','n'),
  INSTRB('León','n'), REGEXP_INSTR('León','n');
```

```
<table>
<thead>
<tr>
<th>default</th>
<th>POSITIONB</th>
<th>pos_chars</th>
<th>pos_oct</th>
<th>INSTR</th>
<th>INSTRB</th>
<th>REGEXP_INSTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
```

**POSITIONB**

Returns an INTEGER value representing the octet location of a specified substring with a string (counting from one).

**Behavior Type**

Immutable

**Syntax**

`POSITIONB ( string, substring )`
Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>(CHAR or VARCHAR) is the string in which to locate the substring</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>(CHAR or VARCHAR) is the substring to locate</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT POSITIONB('straße', 'ße');

POSITIONB
-----------
  5
(1 row)

=> SELECT POSITIONB('étudiant', 'é');

POSITIONB
-----------
  1
(1 row)
```

QUOTE_IDENT

Returns the given string, suitably quoted, to be used as an identifier in a SQL statement string. Quotes are added only if necessary; that is, if the string contains non-identifier characters, is a SQL keyword, such as '1time', 'Next week' and 'Select'. Embedded double quotes are doubled.

Behavior Type

Immutable

Syntax

```sql
QUOTE_IDENT( string )
```

Parameters

| string    | String to quote. |
Notes

- SQL identifiers, such as table and column names, are stored as created, and references to them are resolved using case-insensitive compares. Thus, you do not need to double-quote mixed-case identifiers.
- Vertica quotes all currently-reserved keywords, even those not currently being used.

Examples

Quoted identifiers are case-insensitive, and Vertica does not supply the quotes:

```sql
=> SELECT QUOTE_IDENT('VErTiCa');
QUOTE_IDENT
----------
VErTiCa
(1 row)
=> SELECT QUOTE_IDENT('Vertica database');
QUOTE_IDENT
----------------
"Vertica database"
(1 row)
```

Embedded double quotes are doubled:

```sql
=> SELECT QUOTE_IDENT('Vertica "!" database');
QUOTE_IDENT
----------------
"Vertica "!" database"
(1 row)
```

The following example uses the SQL keyword, SELECT; results are double quoted:

```sql
=> SELECT QUOTE_IDENT('select');
QUOTE_IDENT
---------
"select"
(1 row)
```

QUOTE_LITERAL

Returns the given string, suitably quoted, to be used as a string literal in a SQL statement string. Embedded single quotes and backslashes are doubled.
Behavior Type

Immutable

Syntax

QUOTE_LITERAL ( string )

Parameters

| string | String to convert to a string literal. |

Notes

Vertica recognizes two consecutive single quotes within a string literal as one single quote character. For example, 'You''re here!'. This is the SQL standard representation and is preferred over the form, 'You\'re here!', as backslashes are not parsed as before.

Examples

```sql
=> SELECT QUOTE_LITERAL('You''re here!');
QUOTE_LITERAL
---------------------
'You''re here!'  
(1 row)
```

See Also

- Character String Literals

REPEAT

Replicates a string the specified number of times, and concatenates the replicated values as a single string. The return value can be up to 65000 bytes in length. If the length of string * count is greater than 65000 bytes, Vertica silently truncates the results.
Behavior Type
Immutable

Syntax
REPEAT ('string', count)

Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>The string to repeat, one of the following: CHAR, VARCHAR, BINARY or VARBINARY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>An integer expression that specifies how many times to repeat string.</td>
</tr>
</tbody>
</table>

Examples
The following example repeats vmart three times:

```sql
=> SELECT REPEAT ('vmart', 3);
REPEAT
-------------------
vmartvmartvmart     
(1 row)
```

REPLACE
Replaces all occurrences of characters in a string with another set of characters.

Behavior Type
Immutable

Syntax
REPLACE ('string', 'target', 'replacement')
Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>The string to modify.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The characters in \texttt{string} to replace.</td>
</tr>
<tr>
<td>replacement</td>
<td>The characters to replace \texttt{target}.</td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT REPLACE('Documentation%20Library', '%20', ' ');  
REPLACE
---------------------
Documentation Library
(1 row)
=> SELECT REPLACE('This &That', '&', ';', 'and');  
REPLACE
--------------
This and That
(1 row)
=> SELECT REPLACE('straße', 'ß', 'ss');  
REPLACE
----------
strasse
(1 row)
=> SELECT REPLACE('Customer name: David. The account David owns is overdue. Send David a reminder.', 'David', 'Steve');  
REPLACE
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Customer name: Steve. The account Steve owns is overdue. Send Steve a reminder.
(1 row)
```

**RIGHT**

Returns the specified characters from the right side of a string.

**Behavior Type**

Immutable
Syntax

RIGHT ( string-expr, length )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string-expr</td>
<td>The string expression to return.</td>
</tr>
<tr>
<td>length</td>
<td>An integer value that specifies how many characters to return.</td>
</tr>
</tbody>
</table>

Examples

The following query returns the last three characters of the string 'vertica':

```sql
=> SELECT RIGHT('vertica', 3);
  RIGHT
  ------
  ica
  (1 row)
```

The following query queries date column date_ordered from table store.store_orders_fact. It coerces the dates to strings and extracts the last five characters from each string. It then returns all distinct strings:

```sql
SELECT DISTINCT(
  RIGHT(date_ordered::varchar, 5)) MonthDays
FROM store.store_orders_fact ORDER BY MonthDays;

MonthDays
----------
01-01
01-02
01-03
01-04
01-05
01-06
01-07
01-08
01-09
01-10
02-01
02-02
02-03
...
11-08
11-09
11-10
12-01
```
See Also

SUBSTR

RPAD

Returns a VARCHAR value representing a string of a specific length filled on the right with specific characters.

Behavior Type

Immutable

Syntax

RPAD ( expression, length [ , fill ] )

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>(CHAR OR VARCHAR) specifies the string to fill</td>
</tr>
<tr>
<td>length</td>
<td>(INTEGER) specifies the number of characters to return</td>
</tr>
<tr>
<td>fill</td>
<td>(CHAR OR VARCHAR) specifies the repeating string of characters with which to fill the output string. The default is the space character.</td>
</tr>
</tbody>
</table>
Examples

=> SELECT RPAD('database', 15, 'xzy');
   RPAD
    ---------------
   database xzyxzyx
   (1 row)

If the string is already longer than the specified length it is truncated on the right:

=> SELECT RPAD('database', 6, 'xzy');
   RPAD
    ------
   database
   (1 row)

RTRIM

Returns a VARCHAR value representing a string with trailing blanks removed from the right side (end).

Behavior Type

Immutable

Syntax

RTRIM ( expression [ , characters ] )

Parameters

| expression | (CHAR or VARCHAR) is the string to trim |
| characters | (CHAR or VARCHAR) specifies the characters to remove from the right side of expression. The default is the space character. |
Examples

```sql
=> SELECT RTRIM('trimzzzyyyyyyyyyyyyy', 'xyz');
RTRIM
-----
  trim
(1 row)
```

See Also

- `BTRIM`
- `LTRIM`
- `TRIM`

SHA1

Uses the US Secure Hash Algorithm 1 to calculate the SHA1 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

```
SHA1 ( string )
```

Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>The VARCHAR or VARBINARY string to be calculated.</th>
</tr>
</thead>
</table>
Examples

The following examples calculate the SHA1 hash of the provided strings:

```sql
=> SELECT SHA1('123');
SHA1
-----------------------------
40bd001563085fc35165329ea1ff5c5ecbdbeeef
(1 row)
```

```sql
=> SELECT SHA1('Vertica'::bytea);
SHA1
-----------------------------
e2cffe8d344995c6c301546c4fc5ee152d77c11
(1 row)
```

See Also

- MD5
- SHA224
- SHA256
- SHA384
- SHA512

SHA224

Uses the US Secure Hash Algorithm 2 to calculate the SHA224 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA224 ( string )
Parameters

| string | The VARCHAR or VARBINARY string to be calculated. |

Examples

The following examples calculate the SHA224 hash of the provided strings:

```sql
=> SELECT SHA224('abc');
SHA224('abc')
--------------------------
78d8045d684abd2eece923758f3cd781489df3a48e1278982466017f
(1 row)
```

```sql
=> SELECT SHA224('Vertica':bytea);
SHA224('Vertica':bytea)
------------------------------------------
135ac268f64ff3124aeeebc3cc0af8a29fd600a3be8e29ed97e45e25
(1 row)
```

```sql
=> SELECT sha224('':varbinary) = 'd14a028c2a3a2bc9476102bb288234c415a2b01f828ea62ac5b3e42f' AS "TRUE";
TRUE
------
t
(1 row)
```

See Also

- MD5
- SHA1()
- SHA256()
- SHA384()
- SHA512()
SHA256

Uses the US Secure Hash Algorithm 2 to calculate the SHA256 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA256 ( string )

Parameters

<table>
<thead>
<tr>
<th>string</th>
<th>The VARCHAR or VARBINARY string to be calculated.</th>
</tr>
</thead>
</table>

Examples

The following examples calculate the SHA256 hash of the provided strings:

```sql
=> SELECT SHA256('abc');
   SHA256
       a665a45920422f9d417e4867efdc4fb8a84a1f3ff1fa07e998e86f7f7a27ae3

(1 row)

=> SELECT SHA256('Vertica'::bytea);
   SHA256
       9981b0b7df9f5be06e9e1a7f4ae2336a7868d9ab522b9a6ca6a87cd9ed95ba53

(1 row)

=> SELECT sha256('') = 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855' AS "TRUE";
   TRUE

(1 row)
```
See Also

- MD5
- SHA1
- SHA224
- SHA384
- SHA512

SHA384

Uses the US Secure Hash Algorithm 2 to calculate the SHA384 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA384 ( string )

Parameters

| string | The VARCHAR or VARBINARY string to be calculated. |

Examples

The following examples calculate the SHA384 hash of the provided strings:

```sql
=> SELECT SHA384('123');
SHA384
```
See Also

- MD5
- SHA1
- SHA224
- SHA256
- SHA512

SHA512

Uses the US Secure Hash Algorithm 2 to calculate the SHA512 hash of string. Returns the result as a VARCHAR string in hexadecimal.

Behavior Type

Immutable

Syntax

SHA512 ( string )

Parameters

| string | The VARCHAR or VARBINARY string to be calculated. |
Examples

The following examples calculate the SHA512 hash of the provided strings:

```
=> SELECT SHA512('123');
------------------------------- SHA512
-------------------------------
3c9909afec25354d551dae21590bb26e38d53f2173b8d3dc3eee4c047e7ab1c1eb8b85103e3be7ba613b31bb5c9c36214dc9f14a42fd7a2f6b814856bca5c44c2
(1 row)
```

```
=> SELECT SHA512('Vertica'::bytea);
------------------------------- SHA512
-------------------------------
c4ee2b2d17759226a3897c9c30d7c6df1145c4582849bb5191ee140bce05b83d3d869890cc3619b534f6a97f28a739d8656a5ade6e756b3243ef97d3f00
(1 row)
```

See Also

- MD5
- SHA1
- SHA224
- SHA256
- SHA384

SPACE

Returns the specified number of blank spaces, typically for insertion into a character string.

Behavior Type

Immutable
Syntax

`SPACE(n)`

Parameters

| n   | An integer argument that specifies how many spaces to insert. |

Example

The following example concatenates strings x and y with 10 spaces inserted between them:

```sql
=> SELECT 'x' || SPACE(10) || 'y' AS Ten_spaces;
Ten_spaces
---------
x       y
(1 row)
```

SPLIT_PART

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one).

Behavior Type

Immutable

Syntax

`SPLIT_PART ( string , delimiter , field )`

Parameters

<p>| string | Is the argument string. |</p>
<table>
<thead>
<tr>
<th>delimiter</th>
<th>Is the given delimiter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>(INTEGER) is the number of the part to return.</td>
</tr>
</tbody>
</table>

**Notes**

Use this with the character form of the subfield.

**Examples**

The specified integer of 2 returns the second string, or def.

```sql
=> SELECT SPLIT_PART('abc~@~def~@~ghi', '~@~', 2);
SPLIT_PART
----------
def
(1 row)
```

In the next example, specify 3, which returns the third string, or 789.

```sql
=> SELECT SPLIT_PART('123~|~456~|~789', '~|~', 3);
SPLIT_PART
---------
789
(1 row)
```

The tildes are for readability only. Omitting them returns the same results:

```sql
=> SELECT SPLIT_PART('123|456|789', '|', 3);
SPLIT_PART
---------
789
(1 row)
```

See what happens if you specify an integer that exceeds the number of strings: No results.

```sql
=> SELECT SPLIT_PART('123|456|789', '|', 4);
SPLIT_PART
---------
(1 row)
```

The previous result is not null, it is an empty string.

```sql
=> SELECT SPLIT_PART('123|456|789', '|', 4) IS NULL;
?column?
---------
f
```
If SPLIT_PART had returned NULL, LENGTH would have returned null.

```sql
=> SELECT LENGTH (SPLIT_PART('123|456|789', '|', 4));
LENGTH
-------
 0
(1 row)
```

SPLIT_PARTB

Splits string on the delimiter and returns the location of the beginning of the given field (counting from one). The VARCHAR arguments are treated as octets rather than UTF-8 characters.

Behavior Type

Immutable

Syntax

SPLIT_PARTB (string, delimiter, field)

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>(VARCHAR) Is the argument string.</td>
</tr>
<tr>
<td>delimiter</td>
<td>(VARCHAR) Is the given delimiter.</td>
</tr>
<tr>
<td>field</td>
<td>(INTEGER) is the number of the part to return.</td>
</tr>
</tbody>
</table>

Notes

Use this function with the character form of the subfield.
Examples

The specified integer of 3 returns the third string, or soupçon.

```sql
=> SELECT SPLIT_PARTB('straße~@~café~@~soupçon', '~@~', 3);
SPLIT_PARTB
---------
soupçon
(1 row)
```

The tildes are for readability only. Omitting them returns the same results:

```sql
=> SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 3);
SPLIT_PARTB
---------
soupçon
(1 row)
```

See what happens if you specify an integer that exceeds the number of strings: No results.

```sql
=> SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 4);
SPLIT_PARTB
---------
(1 row)
```

The above result is not null, it is an empty string.

```sql
=> SELECT SPLIT_PARTB('straße @ café @ soupçon', '@', 4) IS NULL;
?column?
---------
f
(1 row)
```

STRPOS

Returns an INTEGER value representing the character location of a specified substring within a string (counting from one).

Behavior Type

Immutable

Syntax

```
STRPOS ( string, substring )
```
Parameters

| string | (CHAR or VARCHAR) is the string in which to locate the substring |
| substring | (CHAR or VARCHAR) is the substring to locate |

Notes

STRPOS is similar to POSITION although POSITION allows finding by characters and by octet. If the string is not found, the return value is zero.

Examples

```sql
=> SELECT STRPOS('abcd', 'c');

<table>
<thead>
<tr>
<th>STRPOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

(1 row)
```

STRPOSB

Returns an INTEGER value representing the location of a specified substring within a string, counting from one, where each octet in the string is counted (as opposed to characters).

Behavior Type

Immutable

Syntax

```
STRPOSB(string, substring)
```
Parameters

| string | (CHAR or VARCHAR) is the string in which to locate the substring |
| substring | (CHAR or VARCHAR) is the substring to locate |

Notes

STRPOSB is identical to POSITIONB except for the order of the arguments.

Examples

```sql
=> SELECT STRPOSB('straße', 'e');
STRPOSB
    -------
     7
(1 row)
=> SELECT STRPOSB('étudiant', 'tud');
STRPOSB
    -------
     3
(1 row)
```

SUBSTR

Returns VARCHAR or VARBINARY value representing a substring of a specified string.

Behavior Type

Immutable

Syntax

```
SUBSTR ( string , position [ , extent ] )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>(CHAR/VARCHAR or BINARY/VARBINARY) is the string from which to extract a substring. If null, Vertica returns no results.</td>
</tr>
<tr>
<td>position</td>
<td>(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by characters). If 0 or negative, Vertica returns no results.</td>
</tr>
<tr>
<td>extent</td>
<td>(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in characters). The default is the end of the string.</td>
</tr>
</tbody>
</table>

Notes

SUBSTR truncates DOUBLE PRECISION input values.

Examples

```sql
=> SELECT SUBSTR('abc':binary(3),1);
   substr
---------
    abc
(1 row)
=> SELECT SUBSTR('123456789', 3, 2);
   substr
------
    34
(1 row)
=> SELECT SUBSTR('123456789', 3);
   substr
------
    3456789
(1 row)
=> SELECT SUBSTR(TO_BITSTRING(HEX_TO_BINARY('0x10')), 2, 2);
   substr
---------
     00
(1 row)
=> SELECT SUBSTR(TO_HEX(10010), 2, 2);
   substr
-------
    71
(1 row)
```
SUBSTRB

Returns an octet value representing the substring of a specified string.

Behavior Type

Immutable

Syntax

SUBSTRB ( string, position [, extent ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>(CHAR/VARCHAR) is the string from which to extract a substring.</td>
</tr>
<tr>
<td>position</td>
<td>(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one in octets).</td>
</tr>
<tr>
<td>extent</td>
<td>(INTEGER or DOUBLE PRECISION) is the length of the substring to extract (in octets). The default is the end of the string.</td>
</tr>
</tbody>
</table>

Notes

- This function treats the multibyte character string as a string of octets (bytes) and uses octet numbers as incoming and outgoing position specifiers and lengths. The strings themselves are type VARCHAR, but they treated as if each octet were a separate character.
- SUBSTRB truncates DOUBLE PRECISION input values.

Examples

```sql
=> SELECT SUBSTRB('soupçon', 5);
SUBSTRB
------------
çon
```
Vertica returns the following error message if you use BINARY/VARBINARY:

```sql
(SELECT SUBSTRB('abc::binary(3)',1);
ERROR: function substrb(binary, int) does not exist, or permission is denied for substrb(binary, int)
HINT: No function matches the given name and argument types. You may need to add explicit type casts.
```

### SUBSTRING

Returns a value representing a substring of the specified string at the given position, given a value, a position, and an optional length. SUBSTRING truncates DOUBLE PRECISION input values.

#### Behavior Type

Immutable if USING OCTETS, stable otherwise.

#### Syntax

```sql
SUBSTRING ( string , position [ , length ]
... [USING {CHARACTERS | OCTETS } ] )
SUBSTRING ( string FROM position [ FOR length ]
... [USING { CHARACTERS | OCTETS } ] )
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>(CHAR/VARCHAR or BINARY/VARBINARY) is the string from which to extract a substring</td>
</tr>
<tr>
<td>position</td>
<td>(INTEGER or DOUBLE PRECISION) is the starting position of the substring (counting from one by either characters or octets). (The default is characters.) If position is greater than the length of the given value, an empty value is returned.</td>
</tr>
</tbody>
</table>
Length

(INTEGER or DOUBLE PRECISION) is the length of the substring to extract in either characters or octets. (The default is characters.) The default is the end of the string. If a length is given the result is at most that many bytes. The maximum length is the length of the given value less the given position. If no length is given or if the given length is greater than the maximum length then the length is set to the maximum length.

USING CHARACTERS | OCTETS

Determines whether the value is expressed in characters (the default) or octets.

Examples

=> SELECT SUBSTRING('abc'::binary(3),1);
  substring
  ---------
  abc
  (1 row)

=> SELECT SUBSTRING('soupçon', 5, 2 USING CHARACTERS);
  substring
  ---------
  ço
  (1 row)

=> SELECT SUBSTRING('soupçon', 5, 2 USING OCTETS);
  substring
  ---------
  ç
  (1 row)

If you use a negative position, then the function starts at a non-existent position. In this example, that means counting eight characters starting at position -4. So the function starts at the empty position -4 and counts five characters, including a position for zero which is also empty. This returns three characters.

=> SELECT SUBSTRING('1234567890', -4, 8);
  substring
  ---------
  123
  (1 row)
TO_BITSTRING

Returns a VARCHAR that represents the given VARBINARY value in bitstring format. This function is the inverse of BITSTRING_TO_BINARY.

Behavior Type

Immutable

Syntax

TO_BITSTRING ( expression )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>The VARCHAR string to process.</th>
</tr>
</thead>
</table>

Examples

```sql
=> SELECT TO_BITSTRING('ab':BINARY(2));
   to_bitstring
-----------------------
0110000101100010
(1 row)

=> SELECT TO_BITSTRING(HEX_TO_BINARY('0x10'));
   to_bitstring
----------------------
00010000
(1 row)

=> SELECT TO_BITSTRING(HEX_TO_BINARY('0xF0'));
   to_bitstring
----------------------
11110000
(1 row)
```

See Also

BITCOUNT
TO_HEX

Returns a VARCHAR or VARBINARY representing the hexadecimal equivalent of a number. This function is the inverse of HEX_TO_BINARY.

Behavior Type

Immutable

Syntax

TO_HEX ( number )

Parameters

| number | An INTEGER or VARBINARY value to convert to hexadecimal. If you supply a VARBINARY argument, the function’s return value is not preceded by 0x. |

Examples

=> SELECT TO_HEX(123456789);
  TO_HEX
  --------
  75bcd15
  (1 row)

For VARBINARY inputs, the returned value is not preceded by 0x. For example:

=> SELECT TO_HEX('ab'::binary(2));
  TO_HEX
  --------
  6162
  (1 row)

TRANSLATE

Replaces individual characters in string_to_replace with other characters.
Behavior Type

Immutable

Syntax

TRANSLATE ( string_to_replace , from_string , to_string );

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_to_replace</td>
<td>String to be translated.</td>
</tr>
<tr>
<td>from_string</td>
<td>Contains characters that should be replaced in string_to_replace.</td>
</tr>
<tr>
<td>to_string</td>
<td>Any character in string_to_replace that matches a character in from_string is replaced by the corresponding character in to_string.</td>
</tr>
</tbody>
</table>

Example

```
=> SELECT TRANSLATE('straße', 'ß', 'ss');
TRANSLATE
----------
strase
(1 row)
```

TRIM

Combines the BTRIM, LTRIM, and RTRIM functions into a single function.

Behavior Type

Immutable

Syntax

TRIM ( [ [ LEADING | TRAILING | BOTH ] characters FROM ] expression )
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEADING</td>
<td>Removes the specified characters from the left side of the string</td>
</tr>
<tr>
<td>TRAILING</td>
<td>Removes the specified characters from the right side of the string</td>
</tr>
<tr>
<td>BOTH</td>
<td>Removes the specified characters from both sides of the string (default)</td>
</tr>
<tr>
<td>characters</td>
<td>(CHAR or VARCHAR) specifies the characters to remove from expression. The default is the space character.</td>
</tr>
<tr>
<td>expression</td>
<td>(CHAR or VARCHAR) is the string to trim</td>
</tr>
</tbody>
</table>

## Examples

```sql
=> SELECT '-' || TRIM(LEADING 'x' FROM 'xxdatabasexx') || '-';
?column?
--------------------
-xxdatabasexx
(1 row)

=> SELECT '-' || TRIM(TRAILING 'x' FROM 'xxdatabasexx') || '-';
?column?
--------------------
-xxdatabase-
(1 row)

=> SELECT '-' || TRIM(BOTH 'x' FROM 'xxdatabasexx') || '-';
?column?
--------------------
-database-
(1 row)

=> SELECT '-' || TRIM('x' FROM 'xxdatabasexx') || '-';
?column?
--------------------
-database-
(1 row)

=> SELECT '-' || TRIM(LEADING FROM ' database ') || '-';
?column?
--------------------
-database -
(1 row)

=> SELECT '-' || TRIM( ' database ') || '-'; ?column?
--------------------
-database-
(1 row)
```
See Also

- BTRIM
- LTRIM
- RTRIM

UPPER

Returns a VARCHAR value containing the argument converted to uppercase letters.

Starting in Release 5.1, this function treats the string argument as a UTF-8 encoded string, rather than depending on the collation setting of the locale (for example, collation=binary) to identify the encoding.

Behavior Type

stable

Syntax

UPPER ( expression )

Parameters

| expression | CHAR or VARCHAR containing the string to convert |

Notes

UPPER is restricted to 32500 octet inputs, since it is possible for the UTF-8 representation of result to double in size.
Examples

```sql
=> SELECT UPPER('AbCdEfG');
  UPPER
  --------
  ABCDEFG
  (1 row)
=> SELECT UPPER('étudiant');
  UPPER
  --------
  ÉTUDIANT
  (1 row)
```

**UPPERB**

Returns a character string with each ASCII character converted to uppercase. Multibyte characters are not converted and are skipped.

**Behavior Type**

Immutable

**Syntax**

`UPPERB ( expression )`

**Parameters**

| expression | (CHAR or VARCHAR) is the string to convert |

**Examples**

In the following example, the multibyte UTF-8 character é is not converted to uppercase:

```sql
=> SELECT UPPERB('étudiant');
  UPPERB
  --------
  étUDIANT
```

Vertica Analytic Database (9.0.x)
V6_ATON

Converts an IPv6 address represented as a character string to a binary string.

Behavior Type

Immutable

Syntax

V6_ATON ( expression )

Parameters

| expression | (VARCHAR) is the string to convert |

Notes

The following syntax converts an IPv6 address represented as the character string A to a binary string B.

V6_ATON trims any spaces from the right of A and calls the Linux function inet_pton.

=> V6_ATON(VARCHAR A) -> VARBINARY(16) B

If A has no colons it is prepended with '::ffff:'. If A is NULL, too long, or if inet_pton returns an error, the result is NULL.
Examples

```sql
=> SELECT V6_ATON('2001:DB8::8:000:200C:417A');
   v6_aton
---------------------------------------------
 \001\15\270\000\000\000\000\010\010\000 \014Az
(1 row)
=> SELECT V6_ATON('1.2.3.4');
   v6_aton
---------------------------------------------
 \000\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
SELECT TO_HEX(V6_ATON('2001:DB8::8:000:200C:417A'));
   to_hex
---------------------------------------------
 20010db8000000000000000200c417a
(1 row)
=> SELECT V6_ATON('::1.2.3.4');
   v6_aton
---------------------------------------------
 \000\000\000\000\000\000\000\000\000\000\000\377\377\001\002\003\004
(1 row)
```

See Also

- [V6_NTOA](#)

V6_NTOA

Converts an IPv6 address represented as varbinary to a character string.

Behavior Type

Immutable

Syntax

```sql
V6_NTOA ( expression )
```
Parameters

| expression | (VARBINARY) is the binary string to convert. |

Notes

The following syntax converts an IPv6 address represented as VARBINARY B to a string A.

V6_NTOA right-pads B to 16 bytes with zeros, if necessary, and calls the Linux function inet_ntop.

=> V6_NTOA(VARBINARY B) -> VARCHAR A

If B is NULL or longer than 16 bytes, the result is NULL.

Vertica automatically converts the form '::ffff:1.2.3.4' to '1.2.3.4'.

Examples

=> SELECT V6_NTOA(' \001\015\270\000\000\000\000\010\010\000 \014Az');
   v6_ntoa
   ------------------------
   2001:db8::8:800:200c:417a
   (1 row)
=> SELECT V6_NTOA(V6_ATON('1.2.3.4'));
   v6_ntoa
   ------
   1.2.3.4
   (1 row)
=> SELECT V6_NTOA(V6_ATON('::1.2.3.4'));
   v6_ntoa
   ---------
   ::1.2.3.4
   (1 row)

See Also

- V6_ATON
V6_SUBNETA

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a binary or alphanumeric IPv6 address.

Behavior Type

Immutable

Syntax

V6_SUBNETA ( expression1, expression2 )

Parameters

| expression1 | (VARBINARY or VARCHAR) is the string to calculate. |
| expression2 | (INTEGER) is the size of the subnet. |

Notes

The following syntax calculates a subnet address in CIDR format from a binary or varchar IPv6 address.

V6_SUBNETA masks a binary IPv6 address B so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared. It then converts to an alphanumeric IPv6 address, appending a slash and N.

=> V6_SUBNETA(BINARY B, INT8 N) -> VARCHAR C

The following syntax calculates a subnet address in CIDR format from an alphanumeric IPv6 address.

=> V6_SUBNETA(VARCHAR A, INT8 N) -> V6_SUBNETA(V6_ATON(A), N) -> VARCHAR C
Examples

```sql
=> SELECT V6_SUBNETA(V6_ATON('2001:db8::8:800:200c:417a'), 28);
    v6_subneta
-----------------
2001:db0::/28
(1 row)
```

See Also

- `V6_SUBNETN`

**V6_SUBNETN**

Calculates a subnet address in CIDR (Classless Inter-Domain Routing) format from a varbinary or alphanumeric IPv6 address.

**Behavior Type**

Immutable

**Syntax**

`V6_SUBNETN ( expression1, expression2 )`

**Parameters**

<table>
<thead>
<tr>
<th>expression1</th>
<th>(VARBINARY or VARCHAR) is the string to calculate.</th>
</tr>
</thead>
</table>

**Notes:**

- `V6_SUBNETN(<VARBINARY>, <INTEGER>)` returns VARBINARY.

  OR

- `V6_SUBNETN(<VARCHAR>, <INTEGER>)` returns VARBINARY, after using V6_ATON to convert the <VARCHAR> string to <VARBINARY>.  


**expression2** (INTEGER) is the size of the subnet.

**Notes**

The following syntax masks a BINARY IPv6 address B so that the N left-most bits of S form a subnet address, while the remaining right-most bits are cleared.

V6_SUBNETN right-pads B to 16 bytes with zeros, if necessary and masks B, preserving its N-bit subnet prefix.

```plaintext
=> V6_SUBNETN(VARBINARY B, INT8 N) -> VARBINARY(16) S
```

If B is NULL or longer than 16 bytes, or if N is not between 0 and 128 inclusive, the result is NULL.

S = \([B]/N\) in Classless Inter-Domain Routing notation (CIDR notation).

The following syntax masks an alphanumeric IPv6 address A so that the N leftmost bits form a subnet address, while the remaining rightmost bits are cleared.

```plaintext
=> V6_SUBNETN(VARCHAR A, INT8 N) -> V6_SUBNETN(V6_ATON(A), N) -> VARBINARY(16) S
```

**Example**

This example returns VARBINARY, after using V6_ATON to convert the VARCHAR string to VARBINARY:

```plaintext
=> SELECT V6_SUBNETN(V6_ATON('2001:db8::8:800:200c:417a'), 28);

v6_subnetn
--------------------------
\001\015\260\000\000\000\000\000\000\000\000\000\000\000\000\000\000
```

**See Also**

- V6_ATON
- V6_SUBNETA

**V6_TYPE**

Characterizes a binary or alphanumeric IPv6 address B as an integer type.
Behavior Type
Immutable

Syntax
V6_TYPE ( expression )

Parameters

| expression | (VARBINARY or VARCHAR) is the type to convert. |

Notes

V6_TYPE(VARBINARY B) returns INT8 T.

=> V6_TYPE(VARCHAR A) -> V6_TYPE(V6_ATON(A)) -> INT8 T

The IPv6 types are defined in the Network Working Group's IP Version 6 Addressing Architecture memo.

- For IPv4, Private-Use is grouped with Link-Local.
- If B is VARBINARY, it is right-padded to 16 bytes with zeros, if necessary.
- If B is NULL or longer than 16 bytes, the result is NULL.

Details

IPv4 (either kind):
Examples

```sql
=> SELECT V6_TYPE(V6_ATON('192.168.2.10'));
  v6_type
--------
   1
(1 row)
```

```sql
=> SELECT V6_TYPE(V6_ATON('2001:db8::8:800:200c:417a'));
  v6_type
--------
    0
(1 row)
```

See Also

- `INET_ATON`
- `IP Version 6 Addressing Architecture`
- `IPv4 Global Unicast Address Assignments`
System Information Functions

These functions provide system information regarding user sessions. A superuser has unrestricted access to all system information, but users can view only information about their own, current sessions.

CURRENT_DATABASE

Returns a VARCHAR value containing the name of the database to which you are connected.

Behavior Type

Immutable

Syntax

CURRENT_DATABASE()

Notes

- The parentheses following the CURRENT_DATABASE function are optional.
- This function is equivalent to DBNAME.

Examples

```
SELECT CURRENT_DATABASE();
CURRENT_DATABASE
VMart
(1 row)
```

The following command returns the same results without the parentheses:

```
SELECT CURRENT_DATABASE;
CURRENT_DATABASE
---------------------
```
CURRENT_SCHEMA

Returns the name of the current schema.

Behavior Type

Stable

Syntax

CURRENT_SCHEMA()

Note: You can call this function without parentheses.

Privileges

None

Examples

The following command returns the name of the current schema:

```sql
=> SELECT CURRENT_SCHEMA();
current_schema
-------------
public
(1 row)
```

The following command returns the same results without the parentheses:

```sql
=> SELECT CURRENT_SCHEMA;
current_schema
-------------
public
(1 row)
```

The following command shows the current schema, listed after the current user, in the search path:
SHOW SEARCH_PATH;

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>search_path</td>
<td>&quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- SET SEARCH_PATH

CURRENT_USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

CURRENT_USER()

Notes

- The CURRENT_USER function does not require parentheses.
- This function is useful for permission checking.
- CURRENT_USER is equivalent to SESSION_USER, USER, and USERNAME.

Examples

```sql
SELECT CURRENT_USER();
CURRENT_USER
---------
dbadmin
(1 row)
```
The following command returns the same results without the parentheses:

```sql
SELECT CURRENT_USER;
CURRENT_USER
-------------
dbadmin
(1 row)
```

**DBNAME (function)**

Returns a VARCHAR value containing the name of the database to which you are connected. DBNAME is equivalent to CURRENT_DATABASE.

**Behavior Type**

Immutable

**Syntax**

DBNAME()

**Examples**

```sql
SELECT DBNAME();
dbname
-------------
vmart
(1 row)
```

**HAS_TABLE_PRIVILEGE**

Indicates whether a user can access a table in a particular way. The function returns a true (t) or false (f) value.

**Behavior Type**

Stable
Syntax

HAS_TABLE_PRIVILEGE ( [ user, ] [ database.]schema.]table, privilege )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Specifies the name or OID of a database user. The default is the CURRENT_USER.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>table</td>
<td>Specifies the name or OID of a table in the logical schema. If necessary, specify the database and schema, as noted above.</td>
</tr>
</tbody>
</table>
| privilege | - SELECT Allows the user to SELECT from any column of the specified table.  
- INSERT Allows the user to INSERT records into the specified table and to use the COPY command to load the table.  
- UPDATE Allows the user to UPDATE records in the specified table.  
- DELETE Allows the user to delete a row from the specified table.  
- REFERENCES Allows the user to create a foreign key constraint (privileges required on both the referencing and referenced tables). |

Privileges

A superuser can check all other user's table privileges.

Users without superuser privileges can use HAS_TABLE_PRIVILEGE to check:
- Any tables they own.
- Tables in a schema to which they have been granted USAGE privileges, and at least one other table privilege, as described in GRANT (Table).

Examples

```sql
SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'SELECT');

<table>
<thead>
<tr>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

SELECT HAS_TABLE_PRIVILEGE('release', 'store.store_dimension', 'INSERT');

<table>
<thead>
<tr>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'UPDATE');

<table>
<thead>
<tr>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

SELECT HAS_TABLE_PRIVILEGE('store.store_dimension', 'REFERENCES');

<table>
<thead>
<tr>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

SELECT HAS_TABLE_PRIVILEGE(45035996273711159, 45035996273711160, 'SELECT');

<table>
<thead>
<tr>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

LIST_ENABLED_CIPHERS

Returns a list of ciphers enabled on your server.

Syntax

```sql
list_enabled_ciphers()
```
Example

=> list_enabled_ciphers()
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA

SESSION_USER

Returns a VARCHAR containing the name of the user who initiated the current database session.

Behavior Type

Stable

Syntax

SESSION_USER()

Notes

- The SESSION_USER function does not require parentheses.
- SESSION_USER is equivalent to CURRENT_USER, USER, and USERNAME.

Examples

SELECT SESSION_USER();
  session_user
  --------------
  dbadmin
(1 row)

The following command returns the same results without the parentheses:
USER

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

USER()

Notes

- The USER function does not require parentheses.
- USER is equivalent to CURRENT_USER, SESSION_USER, and USERNAME.

Examples

```sql
=> SELECT USER();
 current_user
 ---------
 dbadmin
 (1 row)
```

The following command returns the same results without the parentheses:

```sql
=> SELECT USER;
 current_user
 ---------
 dbadmin
 (1 row)
```
USERNAME

Returns a VARCHAR containing the name of the user who initiated the current database connection.

Behavior Type

Stable

Syntax

USERNAME()

Notes

- This function is useful for permission checking.

- USERNAME is equivalent to CURRENT_USER, SESSION_USER and USER.

Examples

```sql
=> SELECT USERNAME();
username
--------
dbadmin
(1 row)
```

VERSION

Returns a VARCHAR containing a Vertica node's version information.

Behavior Type

Stable
Syntax

VERSION()

Examples

```
SELECT VERSION();
```

```
<table>
<thead>
<tr>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertica Analytic Database v4.0.12-20100513010203</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

The parentheses are required. If you omit them, the system returns an error:

```
SELECT VERSION;
```

```
ERROR: column "version" does not exist
```
Timeseries Functions

Timeseries aggregate functions evaluate the values of a given set of variables over time and group those values into a window for analysis and aggregation.

One output row is produced per time slice—or per partition per time slice—if partition expressions are present.

TS_FIRST_VALUE

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, TS_FIRST_VALUE returns the value at the start of the time slice, where an interpolation scheme is applied if the timeslice is missing, in which case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of const (linear).

TS_FIRST_VALUE returns one output row per time slice, or one output row per partition per time slice if partition expressions are specified

Behavior Type

Immutable

Syntax

TS_FIRST_VALUE ( expression [ IGNORE NULLS ] [, [ 'CONST' | 'LINEAR' ] ] )

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>An INTEGER or FLOAT expression on which to aggregate and interpolate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGNORE NULLS</td>
<td>The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See When Time Series Data Contains Nulls in Analyzing Data for details.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'CONST'</th>
<th>'LINEAR'</th>
<th>Specifies the interpolation value as constant or linear:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• CONST (default): New value is interpolated based on previous input records.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• LINEAR: Values are interpolated in a linear slope based on the specified time slice.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Requirements

You must use an ORDER BY clause with a TIMESTAMP column.

Multiple Time Series Aggregate Functions

The same query can call multiple time series aggregate functions. They share the same gap-filling policy as defined by the TIMESERIES Clause; however, each time series aggregate function can specify its own interpolation policy. For example:

```sql
=> SELECT slice_time, symbol, 
   TS_FIRST_VALUE(bid, 'const') fv_c, 
   TS_FIRST_VALUE(bid, 'linear') fv_l, 
   TS_LAST_VALUE(bid, 'const') lv_c 
FROM TickStore 
TIMESERIES slice_time AS '3 seconds' 
OVER(PARTITION BY symbol ORDER BY ts);
```

Examples

See Gap Filling and Interpolation in Analyzing Data.

See Also

- TS_LAST_VALUE
- Time Series Analytics
**TS_LAST_VALUE**

Processes the data that belongs to each time slice. A time series aggregate (TSA) function, TS_LAST_VALUE returns the value at the end of the time slice, where an interpolation scheme is applied if the timeslice is missing. In this case the value is determined by the values corresponding to the previous (and next) timeslices based on the interpolation scheme of const (linear).

TS_LAST_VALUE returns one output row per time slice, or one output row per partition per time slice if partition expressions are specified.

**Behavior Type**

Immutable

**Syntax**

\[
\text{TS}._\text{LAST}._\text{VALUE} \left( \text{expression} \ [\ \text{IGNORE NULLS} \ [\ , \ {\ '\text{CONST'} \ | \ '\text{LINEAR'} } \ ]] \right)
\]

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>expression</em></td>
<td>An INTEGER or FLOAT expression on which to aggregate and interpolate.</td>
</tr>
<tr>
<td>IGNORE NULLS</td>
<td>The IGNORE NULLS behavior changes depending on a CONST or LINEAR interpolation scheme. See <a href="#">When Time Series Data Contains Nulls</a> in Analyzing Data for details.</td>
</tr>
<tr>
<td>'CONST'</td>
<td>'LINEAR'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Requirements

You must use the ORDER BY clause with a TIMESTAMP column.

Multiple Time Series Aggregate Functions

The same query can call multiple time series aggregate functions. They share the same gap-filling policy as defined by the TIMESERIES Clause; however, each time series aggregate function can specify its own interpolation policy. For example:

```sql
=> SELECT slice_time, symbol,
    TS_FIRST_VALUE(bid, 'const') fv_c,
    TS_FIRST_VALUE(bid, 'linear') fv_l,
    TS_LAST_VALUE(bid, 'const') lv_c
FROM TickStore
TIMESERIES slice_time AS '3 seconds'
OVER(PARTITION BY symbol ORDER BY ts);
```

Examples

See Gap Filling and Interpolation in Analyzing Data.

See Also

- TS_FIRST_VALUE
- Time Series Analytics
**URI Encode/Decode Functions**

The functions in this section follow the RFC 3986 standard for percent-encoding a Universal Resource Identifier (URI).

**URI_PERCENT_DECODE**

Decodes a percent-encoded Universal Resource Identifier (URI) according to the RFC 3986 standard.

**Syntax**

```
URI_PERCENT_DECODE (expression)
```

**Behavior Type**

Immutable

**Parameters**

<table>
<thead>
<tr>
<th>expression</th>
<th>(VARCHAR) is the string to convert.</th>
</tr>
</thead>
</table>

**Examples**

The following example invokes uri_percent_decode on the Websites column of the URI table and returns a decoded URI:

```
=> SELECT URI_PERCENT_DECODE(Websites) from URI;
  URI_PERCENT_DECODE
---------------------------------------------------------------
http://www.faqs.org/rfc/rfc3986.html x xj%a%
(1 row)
```

The following example returns the original URI in the Websites column and its decoded version:
URI_PERCENT_ENCODE

Encodes a Universal Resource Identifier (URI) according to the RFC 3986 standard for percent encoding. In addition, for compatibility with older encoders this function converts '+' to space; space is converted to '%20' by uri_percent_encode.

Syntax

URI_PERCENT_ENCODE(expression)

Behavior Type

Immutable

Parameters

description | (VARCHAR) is the string to convert.

Examples

The following example shows how the uri_percent_encode function is invoked on a the Websites column of the URI table and returns an encoded URI:

```sql
=> SELECT URI_PERCENT_ENCODE(Websites) from URI;

<table>
<thead>
<tr>
<th>Websites</th>
<th>URI_PERCENT_ENCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.faqs.org/rfc3986.html+x%20x%6a%25a%25">http://www.faqs.org/rfc3986.html+x%20x%6a%a%</a></td>
<td><a href="http://www.faqs.org/rfc3986.html">http://www.faqs.org/rfc3986.html</a> x j%a%</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>
```

The following example returns the original URI in the Websites column and it's encoded form:
**UUID Functions**

Currently, Vertica provides one function to support UUID data types, **UUID_GENERATE**.

**UUID_GENERATE**

Returns a new universally unique identifier (UUID) that is generated based on high-quality randomness from /dev/urandom.

**Behavior Type**

Volatile

**Syntax**

UUID_GENERATE()

**Example**

```sql
=> CREATE TABLE Customers(
    cust_id UUID DEFAULT(_SELECT uuid_generate()),
    lname VARCHAR(36),
    fname VARCHAR(24));

CREATE TABLE => INSERT INTO Customers VALUES (DEFAULT, 'Kearney', 'Thomas');
    OUTPUT
----------------
   1
(1 row)

=> INSERT INTO Customers VALUES (DEFAULT, 'Pham', 'Duc');
    OUTPUT
----------------
   1
(1 row)
```
=> COMMIT;  
COMMIT  
=> SELECT * FROM Customers;  

<table>
<thead>
<tr>
<th>cust_id</th>
<th>lname</th>
<th>fname</th>
</tr>
</thead>
<tbody>
<tr>
<td>34462732-ed56-4983-8f3b-e735b0c32d50</td>
<td>Kearney</td>
<td>Thomas</td>
</tr>
<tr>
<td>cede66b7-3d29-4da6-b700-871fc0ac57be</td>
<td>Pham</td>
<td>Duc</td>
</tr>
</tbody>
</table>

(2 rows)
Vertica Meta-Functions

Vertica built-in (meta) functions access the internal state of Vertica and are used in SELECT queries with the function name and an argument (where required). These functions are not part of the SQL standard and take the following form:

```
SELECT <meta-function-name>(<args>);
```

**Note:** The query cannot contain other clauses, such as FROM or WHERE.

The behavior type of Vertica meta-functions is immutable.

Alphabetical List of Vertica Meta-Functions

The following list shows all of the Vertica meta-functions in alphabetical order.

Jump to letter: A - B - C - D - E - F - G - H - I - K - L - M - N - P - R - S - V

**ADVANCE_EPOCH**
Manually closes the current epoch and begins a new epoch.

**ALTER_LOCATION_LABEL**
Alters the location label.

**ALTER_LOCATION_SIZE**
Changes the size of the depot location on one node or all of the nodes in the database.

**ALTER_LOCATION_USE**
Alters the type of files that can be stored at the specified storage location.

**ANALYZE_CONSTRAINTS**
Analyzes and reports on constraint violations within the specified scope.

**ANALYZE_CORRELATIONS**
Analyzes the specified tables for pairs of columns that are strongly correlated.

**ANALYZEEXTERNAL_ROW_COUNT**
Calculates the exact number of rows in an external table.

**ANALYZE_STATISTICS**
Collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table.

**ANALYZE_WORKLOAD**
Runs the Workload Analyzer (WLA), a utility that analyzes system information held in system tables.

**AUDIT**
Returns the raw data size (in bytes) of a database, schema, or table as it is counted in an audit of the database size.

**AUDIT_FLEX**
Estimates the ROS size of one or more flexible tables contained in a database, schema, or projection.

**AUDIT_LICENSE_SIZE**
Triggers an immediate audit of the database size to determine if it is in compliance with the raw data storage allowance included in your licenses.

**AUDIT_LICENSE_TERM**
Triggers an immediate audit to determine if the license has expired.

**AWS_GET_CONFIG**
Returns the current Amazon Web Services (AWS) credentials set by AWS_SET_CONFIG or ALTER SESSION.

**AWS_SET_CONFIG**
Gets the values from a table with your Amazon Web Services (AWS) credentials and passes them to session parameters.

**BUILD_FLEXTABLE_VIEW**
creates, or re-creates, a view for a default or user-defined _keys table, ignoring any empty keys.

**CALENDAR_HIERARCHY_DAY**
Specifies to group DATE partition keys into a hierarchy of years, months, and days.

**CANCEL_REBALANCE_CLUSTER**
Stops any rebalance task that is currently in progress or is waiting to execute.

**CANCEL_REFRESH**
Cancels refresh-related internal operations initiated by START_REFRESH and REFRESH.

**CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY**
Changes the run-time priority of an active query.

**CHANGE_RUNTIME_PRIORITY**
Changes the run-time priority of a query that is actively running.

**CLEAR_CACHES**
Clears the internal cache files.

**CLEAR_DATA_COLLECTOR**
Clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the system table DATA_COLLECTOR.
CLEAR_DATA_DEPOT
Deletes the specified data from depots in the database.

CLEAR_HDFS_CACHES
Clears the configuration information copied from HDFS and any cached connections.

CLEAR_OBJECT_STORAGE_POLICY
Removes an existing storage policy.

CLEAR_PROFILING
Clears from memory data for the specified profiling type.

CLEAR_PROJECTION_REFRESHES
Clears information in system table PROJECTION_REFRESHES of projection refresh history.

CLEAR_RESOURCE_REJECTIONS
Clears the content of the RESOURCE_REJECTIONS and DISK_RESOURCE_REJECTIONS system tables.

CLOSE_ALL_RESULTSETS
Closes all result set sessions within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

CLOSE_ALL_SESSIONS
Closes all external sessions except the one that issues this function.

CLOSE_RESULTSET
Closes a specific result set within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

CLOSE_SESSION
Interrupts the specified external session, rolls back the current transaction if any, and closes the socket.

CLOSE_USER_SESSIONS
Stops the session for a user, rolls back any transaction currently running, and closes the connection.

COMPACT_STORAGE
Bundles existing data (.fdb) and index (.pidx) files into the .gt file format.

COMPUTE_FLEXTABLE_KEYS
Computes the virtual columns (keys and values) from the flex table VMap data.

COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW
Combines the functionality of BUILD_FLEXTABLE_VIEW and COMPUTE_FLEXTABLE_KEYS to compute virtual columns (keys) from the VMap data of a flex table and construct a view.

COPY_PARTITION_TO_TABLE
Copies partitions from one table to another.

COPY_TABLE
Copies one table to another.

**CURRENT_SCHEMA**
Returns the name of the current schema.

**DATA_COLLECTOR_HELP**
Returns online usage instructions about the Data Collector, the V_MONITOR.DATA_COLLECTOR system table, and the Data Collector control functions.

**DELETE_COMMUNITY_KEY**
Removes the security key from the database DFS.

**DELETE_TOKENIZER_CONFIG_FILE**
Deletes a tokenizer configuration file.

**DESCRIBE_COMMUNITY_KEY**
Retrieves the value of the security key so you can use it in a query.

**DESIGNER_ADD_DESIGN_QUERIES**
Reads and evaluates queries from an input file, and adds the queries that it accepts to the specified design.

**DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS**
Executes the specified query and evaluates results in the following columns:

**DESIGNER_ADD_DESIGN_QUERY**
Reads and parses the specified query, and if accepted, adds it to the design.

**DESIGNER_ADD_DESIGN_TABLES**
Adds the specified tables to a design.

**DESIGNER_CANCEL_POPULATE_DESIGN**
Cancels population or deployment operation for the specified design if it is currently running.

**DESIGNER_CREATE_DESIGN**
Creates a design with the specified name.

**DESIGNER设计方案_PROJECTION_ENCODINGS**
Analyzes encoding in the specified projections, creates a script to implement encoding recommendations, and optionally deploys the recommendations.

**DESIGNER_DROP_ALL DESIGNS**
Removes all -related schemas associated with the current user.

**DESIGNER_DROP_DESIGN**
Removes the schema associated with the specified design and all its contents.

**DESIGNER_OUTPUT_ALL DESIGN_PROJECTIONS**
Displays the DDL statements that define the design projections to standard output.

**DESIGNER_OUTPUT DEPLOYMENT SCRIPT**
Displays the deployment script for the specified design to standard output.

**DESIGNER_RESET_DESIGN**
Discards all run-specific information of the previous build or deployment of the specified design but keeps its configuration.

**DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY**
Populates the design and creates the design and deployment scripts.

**DESIGNER_SET_ANALYZE_CORRELATIONS_MODE**
Specifies how handles column correlations in a design.

**DESIGNER_SET_DESIGN_KSAFETY**
Sets K-safety for a comprehensive design and stores the K-safety value in the DESIGNS table.

**DESIGNER_SET_DESIGN_TYPE**
Specifies whether should create a comprehensive or incremental design.

**DESIGNER_SET_OPTIMIZATION_OBJECTIVE**
Valid only for comprehensive database designs, specifies the optimization objective uses.

**DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS**
Specifies whether a design can include unsegmented projections.

**DESIGNER_WAIT_FOR_DESIGN**
Waits for completion of operations that are populating and deploying the design.

**DISABLE_DUPLICATE_KEY_ERROR**
Disables error messaging when finds duplicate primary or unique key values at run time (for use with key constraints that are not automatically enabled).

**DISABLE_ELASTIC_CLUSTER**
Disables elastic cluster scaling, which prevents from bundling data into chunks that are easily transportable to other nodes when performing cluster resizing.

**DISABLE_LOCAL_SEGMENTS**
Disables local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes.

**DISABLE_PROFILING**
Disables profiling for the specified profiling type.

**DISPLAY_LICENSE**
Returns the terms of your license.

**DO_TM_TASK**
Runs a Tuple Mover operation on the specified table or projection and commits any current transaction.

**DROP_EXTERNAL_ROW_COUNT**
Removes external table row count statistics compiled by ANALYZE_EXTERNAL_ROW_COUNT.
DROP_LICENSE
   Drops a license key from the global catalog.

DROP_LOCATION
   Removes the specified storage location.

DROP_PARTITIONS
   Drops the specified table partition keys.

DROP_STATISTICS
   Removes statistical data on database projections previously generated by ANALYZE_STATISTICS.

DUMP_CATALOG
   Returns an internal representation of the catalog.

DUMP_LOCKTABLE
   Returns information about deadlocked clients and the resources they are waiting for.

DUMP_PARTITION_KEYS
   Dumps the partition keys of all projections in the system.

DUMP_PROJECTION_PARTITION_KEYS
   Dumps the partition keys of the specified projection.

DUMP_TABLE_PARTITION_KEYS
   Dumps the partition keys of all projections for the specified table.

EMPTYMAP
   Constructs a new VMap with one row but without keys or data.

ENABLE_ELASTIC_CLUSTER
   Enables elastic cluster scaling, which makes enlarging or reducing the size of your database cluster more efficient by segmenting a node's data into chunks that can be easily moved to other hosts.

ENABLE_LOCAL_SEGMENTS
   Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes.

ENABLE_PROFILING
   Enables profiling for the specified profiling type.

ENFORCE_OBJECT_STORAGE_POLICY
   Applies object storage policies immediately, instead of waiting for the Tuple Mover to perform the next moveout.

EVALUATE_DELETE_PERFORMANCE
   Evaluates projections for potential DELETE performance issues.

EXPORT_CATALOG
   Generates a SQL script for recreating a physical schema design on another cluster.
EXPORT_DIRECTED_QUERIES
Generates SQL for creating directed queries from a set of input queries, and writes the SQL to the specified file or to standard output.

EXPORT_OBJECTS
Generates a SQL script you can use to recreate non-virtual catalog objects on another cluster.

EXPORT_STATISTICS
Generates database statistics in XML format from data previously collected by ANALYZE_STATISTICS.

EXPORT_TABLES
Generates a SQL script that can be used to recreate a logical schema—schemas, tables, constraints, and views—on another cluster.

FINISH_FETCHING_FILES
Fetches all the files that are queued for fetching from the communal storage.

FLUSH_DATA_COLLECTOR
Waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the log with the disk storage.

FLUSH_REAPER_QUEUE
Deletes all of the data marked for deletion in the database.

GET_AHM_EPOCH
Returns the number of the epoch in which the Ancient History Mark is located.

GET_AHM_TIME
Returns a TIMESTAMP value representing the Ancient History Mark.

GET_AUDIT_TIME
Reports the time when the automatic audit of database size occurs.

GET_CLIENT_LABEL
Returns the client connection label for the current session.

GET_COMPLIANCE_STATUS
Displays whether your database is in compliance with your license agreement.

GET_CURRENT_EPOCH
The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.

GET_DATA_COLLECTOR_POLICY
Retrieves a brief statement about the retention policy for the specified component.

GET_LAST_GOOD_EPOCH
Returns the last good epoch number.

GET_NUM_ACCEPTED_ROWS
Returns the number of rows loaded into the database for the last completed load for the current session.

GET_NUM_REJECTED_ROWS
Returns the number of rows that were rejected during the last completed load for the current session.

GET_PROJECTIONS
Returns the following information about projections of the specified anchor table:

GET_PROJECTION_STATUS
Returns information relevant to the status of a projection:

GET_TOKENIZER_PARAMETER
Returns the configuration parameter for a given tokenizer.

HAS_ROLE
Indicates, with a Boolean value, whether a role has been assigned to a user.

HIBERNATE
In , uploads transaction logs, checkpoints, logs, and DC tables to AWS S3.

IDOL_CHECK_ACL
Checks the access control list to verify that the user has permissions to access the data in the IDOL flex table and any views created from the table.

IMPORT_DIRECTED_QUERIES
Imports to the database catalog directed queries from a SQL file that was generated by EXPORT_DIRECTED_QUERIES.

IMPORT_STATISTICS
Imports statistics from the XML file that was generated by EXPORT_STATISTICS.

INSTALL_COMMUNITY_KEY
Stores the Connector Framework Service security key in the Distributed File System (DFS).

INSTALL_LICENSE
Installs the license key in the global catalog.

INTERRUPT_STATEMENT
Interrupts the specified statement in a user session, rolls back the current transaction, and writes a success or failure message to the log file.

KERBEROS_CONFIG_CHECK
Tests the Kerberos configuration of a cluster.

KERBEROS_HDFS_CONFIG_CHECK
Tests the Kerberos configuration of a cluster that uses HDFS.

LAST_INSERT_ID
Returns the last value of an AUTO_INCREMENT / IDENTITY column.
LDAP_Link_Sync_Start
   Begins the synchronization between the LDAP server and immediately rather than waiting for the interval set in LDAPLinkInterval.

Make_Ahm_Now
   Sets the Ancient History Mark (AHM) to the greatest allowable value.

MapAggregate
   Returns a LONG VARBINARY VMap with keys and value pairs supplied from two VARCHAR input columns of an existing columnar table.

MapContainsKey
   Determines whether a VMap contains a virtual column (key).

MapContainsValue
   Determines whether a VMap contains a specific value.

MapDelimitedExtractor
   Extracts data with a delimiter character, and other optional arguments, returning a single VMap value.

MapItems
   Returns information about items in a VMap.

MapJsonExtractor
   Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements.

MapKeys
   Returns the virtual columns (and values) present in any VMap data.

MapKeysInfo
   Returns virtual column information from a given map.

MapLookup
   Returns single-key values from VMAP data.

MapPut
   Accepts a VMap and one or more key/value pairs and returns a new VMap with the key/value pairs added.

MapRegexExtractor
   Extracts data from a regular expression and returns the results as a VMap.

MapSize
   Returns the number of virtual columns present in any VMap data.

MapToString
   Recursively builds a string representation VMap data, including nested JSON maps.

MapValues
   Returns a string representation of the top-level values from a VMap.
MAPVERSION
   Returns the version or invalidity of any map data.

MARK_DESIGN_KSAFE
   Enables or disables high availability in your environment, in case of a failure.

MATERIALIZE_FLEXTABLE_COLUMNS
   Materializes virtual columns listed as key_names in the flextable_keys table you compute using either COMPUTE_FLEXTABLE_KEYS or COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW.

MEASURE_LOCATION_PERFORMANCE
   Measures disk performance for the location specified.

MOVE_PARTITIONS_TO_TABLE
   Moves partitions from one table to another.

MOVE RETIRED LOCATION DATA
   Moves all data from either a single retired storage location or all retired storage locations in the database.

MOVE STATEMENT TO RESOURCE POOL
   Attempts to move the specified query to the specified target pool.

NOTIFY
   Specifies the text message to include with a notification.

PARTITION_PROJECTION
   Splits ROS containers for a specified projection.

PARTITION_TABLE
   Invokes the Tuple Mover to reorganize ROS storage containers as needed to conform with the current partitioning policy.

PURGE
   Permanently removes delete vectors from ROS storage containers so disk space can be reused.

PURGE_PARTITION
   Purges a table partition of deleted rows.

PURGE_PROJECTION
   Permanently removes deleted data from physical storage so disk space can be reused.

PURGE_TABLE
   Permanently removes deleted data from physical storage so disk space can be reused.

READ_CONFIG_FILE
   Reads and returns the key-value pairs of all the parameters of a given tokenizer.

REALIGN_CONTROL_NODES
Chooses control nodes (spread hosts) from all cluster nodes and assigns the rest of the nodes in the cluster to a control node.

**REBALANCE_CLUSTER**
Rebalances the database cluster synchronously as a session foreground task.

**REBALANCE_SHARDS**
Rebalances shard assignments across the cluster.

**REBALANCE_TABLE**
Synchronously rebalances data in the specified table.

**REENABLE_DUPLICATE_KEY_ERROR**
Restores the default behavior of error reporting by reversing the effects of DISABLE_DUPLICATE_KEY_ERROR.

**REFRESH**
Synchronously refreshes one or more table projections in the foreground, and updates system table PROJECTION_REFRESHES.

**REFRESH_COLUMNS**
Refreshes table columns that are defined with the constraint SET USING.

**RELEASE_ALL_JVM_MEMORY**
Forces all sessions to release the memory consumed by their Java Virtual Machines (JVM).

**RELEASE_JVM_MEMORY**
Terminates a Java Virtual Machine (JVM), making available the memory the JVM was using.

**RELEASE_SYSTEM_TABLES_ACCESS**
Opens access to non-superuser-only tables that are not accessible during lockdown.

**RELOAD_SPREAD**
Updates cluster changes to the catalog's spread configuration file.

**RESERVE_SESSION_RESOURCE**
Reserves memory resources from the general resource pool for the exclusive use of the backup and restore process.

**RESET_LOAD_BALANCE_POLICY**
Resets the counter each host in the cluster maintains, to track which host it will refer a client to when the native connection load balancing scheme is set to ROUNDROBIN.

**RESET_SESSION**
Applies your default connection string configuration settings to your current session.

**RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW**
Restores the _keys table and the _view.

**RESTORE_LOCATION**
Restores a storage location that was previously retired with RETIRE_LOCATION.
RESTRICT_SYSTEM_TABLES_ACCESS
- Restricts access to non-superuser-only tables that are not accessible during lockdown.

RETIRE_LOCATION
- Makes the specified storage location inactive.

S3
- Identifies the source location of files in an Amazon S3 bucket.

S3EXPORT
- Exports data to an Amazon S3 bucket from your cluster.

SET_AHM_EPOCH
- Sets the Ancient History Mark (AHM) to the specified epoch.

SET_AHM_TIME
- Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node.

SET_AUDIT_TIME
- Sets the time that performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your license.

SET_CLIENT_LABEL
- Assigns a label to a client connection for the current session.

SET_CONFIG_PARAMETER
- Specifies the value of a configuration parameter at the database level, or for a specific node.

SET_CONTROL_SET_SIZE
- Specifies the number of cluster nodes on which to deploy control messaging (spread).

SET_DATA_COLLECTOR_POLICY
- Sets a size restraint (memory and disk space in kilobytes) for the specified Data Collector table on all nodes.

SET_DATA_COLLECTOR_TIME_POLICY
- Sets a time capacity for individual Data Collector tables on all nodes.

SET_LOAD_BALANCE_POLICY
- Sets how native connection load balancing chooses a host to handle a client connection.

SET_LOCATION_PERFORMANCE
- Sets disk performance for the location specified.

SET_OBJECT_STORAGE_POLICY
- Creates or changes an object storage policy by associating a database object with a labeled storage location.

SET_SCALING_FACTOR
- Sets the scaling factor that determines the number of storage containers used when rebalancing the database and when using local data segmentation is enabled.
**SET_TOKENIZER_PARAMETER**
Configures the tokenizer parameters.

**SHOW_PROFILES_CONFIG**
Shows whether profiling is enabled.

**SHUTDOWN**
Forces a database to shut down, even if there are users connected.

**SLEEP**
Waits a specified number of seconds before executing another statement or command.

**START_REBALANCE_CLUSTER**
Asynchronously rebalances the database cluster as a background task.

**START_REFRESH**
Refreshes projections in the current schema with the latest data of their respective anchor tables.

**SWAP_PARTITIONS_BETWEEN_TABLES**
Swaps partitions between two tables.

**SYNC_CATALOG**
Immediately synchronizes the catalog to shared storage to allow revive as of the current catalog version in the case of an imminent crash.

**SYNC_WITH_HCATALOG_SCHEMA**
Copies the structure of a Hive database schema available through the HCatalog Connector to a schema.

**SYNC_WITH_HCATALOG_SCHEMA_TABLE**
Copies the structure of a single table in a Hive database schema available through the HCatalog Connector to a table.

**VALIDATE_STATISTICS**
Validates statistics in the XML file generated by the EXPORT_STATISTICS command.

**VERIFY_HADOOP_CONF_DIR**
Verifies that the Hadoop configuration that is used to access HDFS is valid on all nodes.

### AWS Library Functions

This section contains the functions associated with the Vertica library for Amazon Web Services (AWS).
**AWS_GET_CONFIG**

Returns the current Amazon Web Services (AWS) credentials set by `AWS_SET_CONFIG` or `ALTER SESSION`.

**Syntax**

```sql
AWS_GET_CONFIG( 'parameter' )
```

**Parameter**

- `aws_id`
- `aws_secret`
- `aws_region`
- `aws_ca_path`
- `aws_ca_bundle`
- `aws_proxy`
- `aws_verbose`
- `aws_max_send_speed`
- `aws_max_recv_speed`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aws_id</code></td>
<td>Retrieves the value for the 20-character AWS access key used to authenticate your account</td>
</tr>
<tr>
<td><code>aws_secret</code></td>
<td>Retrieves the value for the 40-character AWS secret access key used to authenticate your account</td>
</tr>
<tr>
<td><code>aws_region</code></td>
<td>Retrieves the region where your AWS bucket is located. See the AWS Documentation for the full list of values. Default value: <code>us-east-1</code></td>
</tr>
<tr>
<td><code>aws_ca_path</code></td>
<td>Retrieves the path Vertica uses to look up SSL server certificates.</td>
</tr>
<tr>
<td><code>aws_ca_bundle</code></td>
<td>Retrieves the path Vertica uses to look up an SSL server certificate bundle.</td>
</tr>
<tr>
<td><code>aws_proxy</code></td>
<td>A string value that lets you set an HTTP/HTTPS proxy for the AWS library.</td>
</tr>
<tr>
<td><code>aws_verbose</code></td>
<td>When enabled, logs libcurl debug messages to <code>dbLog</code>.</td>
</tr>
<tr>
<td><code>aws_max_send_speed</code></td>
<td>Retrieves the value for the maximum transfer speed when sending data to AWS S3, in bytes per second.</td>
</tr>
</tbody>
</table>
**aws_max_recv_speed**

Retrieves the value for the maximum transfer speed for receiving data to AWS S3, in bytes per second.

### Examples

This example retrieves a stored AWS access key in a session.

```sql
=> SELECT AWS_GET_CONFIG('aws_id');
aws_get_config
AKABCOEXAMPLEPKPXYZQ
(1 row)
```

### See Also

- [AWS Library Parameters](#)
- [Vertica AWS Library](#)

### AWS_SET_CONFIG

Gets the values from a table with your Amazon Web Services (AWS) credentials and passes them to session parameters.

### Syntax

```sql
AWS_SET_CONFIG( 'parameter' , 'value' )
```

**parameter**

- `aws_id`
- `aws_secret`
- `aws_region`
- `aws_ca_path`
- `aws_ca_bundle`
- `aws_proxy`
- `aws_verbose`
- `aws_max_send_speed`
- `aws_max_recv_speed`

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aws_id</code></td>
<td>Specifies the 20-character AWS access key used to authenticate your account.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>aws_secret</td>
<td>Specifies the 40-character AWS secret access key used to authenticate your account.</td>
</tr>
<tr>
<td>aws_region</td>
<td>Specifies the region where your AWS bucket is located. See the <a href="https://docs.aws.amazon.com/">AWS Documentation</a> for the full list of values. You can configure aws_region with only one region. To access buckets in multiple regions, reset the parameter each time you change regions. <strong>Default value</strong>: us-east-1</td>
</tr>
<tr>
<td>aws_ca_path</td>
<td>The path Vertica uses to look up SSL server certificates. <strong>Default value</strong>: System-dependent</td>
</tr>
<tr>
<td>aws_ca_bundle</td>
<td>The path Vertica uses to look up an SSL server certificate bundle. <strong>Default value</strong>: System-dependent</td>
</tr>
<tr>
<td>aws_proxy</td>
<td>A string value that lets you set an HTTP/HTTPS proxy for the AWS library.</td>
</tr>
<tr>
<td>aws_verbose</td>
<td>When enabled, logs libcurl debug messages to dbLog. <strong>Default value</strong>: false</td>
</tr>
<tr>
<td>aws_max_send_speed</td>
<td>Sets the maximum transfer speed for sending data to AWS S3, in bytes per second. <strong>Default value</strong>: unlimited</td>
</tr>
<tr>
<td>aws_max_recv_speed</td>
<td>Sets the maximum transfer speed when receiving data to AWS S3, in bytes per second. <strong>Default value</strong>: unlimited</td>
</tr>
</tbody>
</table>

**Examples**

Set the `aws_region` parameter to `us-east-1`.

```
=> SELECT AWS_SET_CONFIG ('aws_region', 'us-east-1');
AWS_SET_CONFIG
------------------
aws_region
(1 row)
```

Configure session parameters for an AWS access key and secret access key with credentials in table keychain:

---

**Vertica Analytic Database (9.0.x)**
== SELECT AWS_SET_CONFIG('aws_id', accesskey),
    AWS_SET_CONFIG('aws_secret', secretaccesskey)
FROM keychain;
AWS_SET_CONFIG | AWS_SET_CONFIG
-----------------+-------------------
aws_id           | aws_secret
(1 row)

See Also

- AWS_GET_CONFIG
- AWS Library Parameters
- Vertica AWS Library

S3

Identifies the source location of files in an Amazon S3 bucket. Use the S3 function in conjunction with a COPY statement to import data into a Vertica cluster from an S3 object.

Syntax

One of the following:

S3( url='object-url'[,..., delimiter='char'] )
S3( bucket='object-url' )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url [, delimiter='char']</td>
<td>Specifies the URLs of one or more S3 objects to import, either the HTTPS URL or the S3 URL. URLs should contain only ASCII characters, 0x01 to 0x7F. If you specify multiple URLs, you can optionally qualify this parameter by specifying a delimiter character, by default</td>
</tr>
<tr>
<td></td>
<td>: (colon)</td>
</tr>
<tr>
<td></td>
<td>- (hyphen)</td>
</tr>
<tr>
<td></td>
<td>, (comma)</td>
</tr>
</tbody>
</table>
bucket
URL of the bucket when importing multiple files using glob expansion.

Privileges
Write privileges on the table you are copying to.

Examples
The following statement specifies to import multiple files. Object URLs are delimited by vertical bars, the default delimiter:

```
=> COPY exampleTable SOURCE s3('s3://exampleBucket/object1|s3://exampleBucket/object2');
```

Import all files in a bucket using glob expansion:

```
=> COPY table1 WITH SOURCE S3('s3://exampleBucket/*');

=> COPY exampleTable SOURCE S3('s3://exampleBucket/');

=> COPY exampleTable SOURCE S3('s3://exampleBucket/');
```

See Also
- AWS_SET_CONFIG
- AWS_GET_CONFIG
- AWS Library Parameters
- Vertica AWS Library

S3EXPORT
Exports data to an Amazon S3 bucket from your Vertica cluster.

If your bucket name contains a period, you must specify a file extension in your export syntax or enable the prepend_hash parameter in s3export.
Syntax

S3EXPORT( expression USING PARAMETERS { parameter=setting} [,...]

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>Specifies the source of the export operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the S3 bucket and object base name. This value can be either the HTTPS URL or the S3 URL. URL length is limited to a maximum of 236 characters. URLs should contain only ASCII characters, 0x01 to 0x7F.</td>
</tr>
<tr>
<td>delimiter</td>
<td>Specifies the column delimiter character. Default value:</td>
</tr>
<tr>
<td>multipart</td>
<td>Deprecated; has no effect.</td>
</tr>
<tr>
<td>chunksize</td>
<td>Specifies in bytes the maximum size of each part. Valid settings can range between 5 MB and 5 GB. The maximum number of chunks allowed in an export is 10000. Default value: 10485760</td>
</tr>
<tr>
<td>record_terminator</td>
<td>Specifies what character marks the end of a record. Default value: \n</td>
</tr>
<tr>
<td>from_charset</td>
<td>Specifies the character set in which your data is currently encoded.</td>
</tr>
<tr>
<td>to_charset</td>
<td>Specifies the character set in which you want to encode your export.</td>
</tr>
<tr>
<td>prepend_hash</td>
<td>Prepends the unique hash values assigned to exported objects instead of the standard appendation. If your S3 bucket contains a period in its path, set the prepend_hash parameter to true. Default value: false</td>
</tr>
</tbody>
</table>

from_charset and to_charset values are system-dependent. Refer to your operating system documentation for more details.
Examples

Export column1 data from exampleTable:

```sql
=> SELECT s3export(column1 USING PARAMETERS
   url='s3://exampleBucket/object',
   delimiter=',',
   chunksize='10485760',
   record_terminator='\n',
   from_charset='ASCII',
   to_charset='UTF-8',
   prepend_hash='true')
OVER () FROM exampleTable;
```

See Also

- AWS_SET_CONFIG
- AWS_GET_CONFIG
- AWS Library Parameters
- Vertica AWS Library

Catalog Management Functions

This section contains catalog management functions specific to Vertica.

DROP_LICENSE

Drops a license key from the global catalog.

Syntax

```
DROP_LICENSE( 'license-name' )
```

Parameters

| license-name | The name of the license to drop. Use the name (or long license key) in the NAME column of system table LICENSES. |
Privileges

Superuser

Examples

```sql
=> SELECT DROP_LICENSE('9b2d81e2-aab1-4cfb-bc07-fa9a696e8f5e');
```

See Also

Managing Licenses

**DUMP_CATALOG**

Returns an internal representation of the Vertica catalog. This function is used for diagnostic purposes.

DUMP_CATALOG returns only the objects that are visible to the user.

Syntax

```
DUMP_CATALOG()
```

Privileges

None

Examples

The following query obtains an internal representation of the Vertica catalog:

```sql
=> SELECT DUMP_CATALOG();
```

The output is written to the specified file:

```
\o /tmp/catalog.txtSELECT DUMP_CATALOG();
\o
```
EXPORT_CATALOG

Generates a SQL script for recreating a physical schema design on another cluster. This function always attempts to recreate projection statements with KSAFE clauses, if they exist in the original definitions, or OFFSET clauses if they do not.

Syntax

```
EXPORT_CATALOG ( ['destination'] [, 'scope' ] )
```

Parameters

If you omit all parameters, EXPORT_CATALOG exports to standard output all schemas, tables, constraints, views, and projections to which the user has access.

<table>
<thead>
<tr>
<th><code>destination</code></th>
<th>Specifies where to send output, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• An empty string (' ') writes the script to standard output.</td>
</tr>
<tr>
<td></td>
<td>• The path and name of a SQL output file. This option is valid only for superusers. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>scope</code></th>
<th>Determines what to export:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• DESIGN (default): Exports schemas, tables, constraints, views, projections, and SQL macros to which the user has access. See also EXPORT_OBJECTS.</td>
</tr>
<tr>
<td></td>
<td>• DESIGN_ALL: Exports all design objects plus system objects created in Database Designer—for example, design contexts and their tables. Exported objects are those to which the user has access.</td>
</tr>
<tr>
<td></td>
<td>• TABLES: Exports all tables and constraints for which the user has access. See also EXPORT_TABLES.</td>
</tr>
<tr>
<td></td>
<td>• DIRECTED_QUERIES: Exports all directed queries that are stored in the database. For more information, see Managing Directed Queries in the Administrator's Guide.</td>
</tr>
</tbody>
</table>
Privileges
None

Example
See Exporting the Catalog.

See Also
- EXPORT_OBJECTS
- EXPORT_TABLES
- Exporting the Catalog

EXPORT_OBJECTS
Generates a SQL script you can use to recreate non-virtual catalog objects on another cluster. The following requirements apply:

- EXPORT_OBJECTS only exports objects to which the user has access.
- EXPORT_OBJECTS exports objects in order dependency for correct recreation. When you run the script on another cluster, Vertica creates all referenced objects before their dependent objects.
- EXPORT_OBJECTS always tries to recreate projection statements with their KSAFE clause, if any, otherwise with their OFFSET clause.

Syntax
EXPORT_OBJECTS( 'destination' [, 'scope' ] [, 'ksafe' ] )

Parameters

<table>
<thead>
<tr>
<th>destination</th>
<th>Specifies where to send output, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An empty string ('') writes the script to standard output.</td>
</tr>
</tbody>
</table>
**The path and name of a SQL output file. This option is valid only for superusers.** If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.

| **scope** | Specifies one or more objects to export, as follows:
| [database.]schema[.object][,...] | If set to an empty string, Vertica exports all objects to which the user has access, including constraints.
| If you specify a schema, Vertica exports all objects in that schema.
| If you specify a database, it must be the current database. |
| **ksafe** | Specifies whether to include a MARK\_DESIGN\_KSAFE statement in the generated script with the correct K-safe value for the database:
| • true (default): Include the MARK\_DESIGN\_KSAFE statement at the end of the output script.
| • false: Omit the MARK\_DESIGN\_KSAFE statement from the script. |

**Privileges**

None

**Example**

See Exporting Objects.

**See Also**

- EXPORT\_CATALOG
- EXPORT\_TABLES

**EXPORT\_TABLES**

Generates a SQL script that can be used to recreate a logical schema—schemas, tables, constraints, and views—on another cluster. EXPORT\_OBJECTS only exports objects to which
the user has access.

Syntax

\[
\text{EXPORT\_TABLES}(\ ['\text{destination}\',\ ',\ '\text{scope}\'\ ])
\]

Parameters

<table>
<thead>
<tr>
<th>\textit{destination}</th>
<th>Specifies where to send output, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• An empty string ('') writes the script to standard output.</td>
</tr>
<tr>
<td></td>
<td>• The path and name of a SQL output file. This option is valid only for superusers. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\textit{scope}</th>
<th>Specifies one or more tables to export, as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{[database.]schema[.table][,...]}</td>
<td></td>
</tr>
<tr>
<td>If set to an empty string, Vertica exports all non-virtual table objects to which the user has access, including table schemas, sequences, and constraints.</td>
<td></td>
</tr>
<tr>
<td>If you specify a schema, Vertica exports all non-virtual table objects in that schema.</td>
<td></td>
</tr>
<tr>
<td>If you specify a database, it must be the current database.</td>
<td></td>
</tr>
</tbody>
</table>

Privileges

None

Example

See Exporting Tables.

The following example exports to standard output all tables in the \texttt{store} schema:

\[
\begin{verbatim}
\text{SELECT EXPORT\_TABLES(\''\',\'store\');}
\end{verbatim}
\]
CREATE TABLE store.store_dimension
(
    store_key int NOT NULL,
    store_name varchar(64),
    store_number int,
    store_address varchar(256),
    store_city varchar(64),
    store_state char(2),
    store_region varchar(64),
    floor_plan_type varchar(32),
    photo_processing_type varchar(32),
    financial_service_type varchar(32),
    selling_square_footage int,
    total_square_footage int,
    first_open_date date,
    last_remodel_date date,
    number_of_employees int,
    annual_shrinkage int,
    foot_traffic int,
    monthly_rent_cost int
);

ALTER TABLE store.store_dimension ADD CONSTRAINT C_PRIMARY PRIMARY KEY (store_key) DISABLE;

CREATE TABLE store.store_sales_fact
(
    date_key int NOT NULL,
    product_key int NOT NULL,
    product_version int NOT NULL,
    store_key int NOT NULL,
    promotion_key int NOT NULL,
    customer_key int NOT NULL,
    employee_key int NOT NULL,
    pos_transaction_number int NOT NULL,
    sales_quantity int,
    sales_dollar_amount int,
    cost_dollar_amount int,
    gross_profit_dollar_amount int,
    transaction_type varchar(16),
    transaction_time time,
    tender_type varchar(8)
);

CREATE TABLE store.store_orders_fact
(
    product_key int NOT NULL,
    product_version int NOT NULL,
    store_key int NOT NULL,
    vendor_key int NOT NULL,
    employee_key int NOT NULL,
    order_number int NOT NULL,
    date_ordered date,
    date_shipped date,
    expected_delivery_date date,
    date_delivered date,
    quantity_ordered int,
    quantity_delivered int,
    shipper_name varchar(32),
    unit_price int,
    shipping_cost int,
total_order_cost int,
quantity_in_stock int,
reorder_level int,
overstock_ceiling int
);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_date FOREIGN KEY (date_key) references public.date_dimension (date_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_product FOREIGN KEY (product_key, product_version) references public.product_dimension (product_key, product_version);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_store FOREIGN KEY (store_key) references store.store_dimension (store_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_promotion FOREIGN KEY (promotion_key) references public.promotion_dimension (promotion_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_customer FOREIGN KEY (customer_key) references public.customer_dimension (customer_key);
ALTER TABLE store.store_sales_fact ADD CONSTRAINT fk_store_sales_employee FOREIGN KEY (employee_key) references public.employee_dimension (employee_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_product FOREIGN KEY (product_key, product_version) references public.product_dimension (product_key, product_version);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_store FOREIGN KEY (store_key) references store.store_dimension (store_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_vendor FOREIGN KEY (vendor_key) references public.vendor_dimension (vendor_key);
ALTER TABLE store.store_orders_fact ADD CONSTRAINT fk_store_orders_employee FOREIGN KEY (employee_key) references public.employee_dimension (employee_key);

(1 row)

See Also

- EXPORT_CATALOG
- EXPORT_OBJECTS

INSTALL_LICENSE

Installs the license key in the global catalog.

Syntax

INSTALL_LICENSE( 'filename' )

Parameters

| filename | The absolute path name of a valid license file. |
Privileges
Superuser

Examples

```sql
=> SELECT INSTALL_LICENSE('/tmp/vlicense.dat');
```

See Also
Managing Licenses

**MARK DESIGN KSAFE**

Enables or disables high availability in your environment, in case of a failure. Before enabling recovery, **MARK DESIGN KSAFE** queries the catalog to determine whether a cluster's physical schema design meets the following requirements:

- Small, unsegmented tables are replicated on all nodes.
- Large table superprojections are segmented with each segment on a different node.
- Each large table projection has at least one buddy projection for K-safety=1 (or two buddy projections for K-safety=2).

  Buddy projections are also segmented across database nodes, but the distribution is modified so segments that contain the same data are distributed to different nodes. See [High Availability With Projections](#) in Vertica Concepts.

**MARK DESIGN KSAFE** does not change the physical schema.

**Syntax**

```
MARK DESIGN KSAFE ( k )
```

**Parameters**

<table>
<thead>
<tr>
<th>k</th>
<th>Specifies the level of K-safety, one of the following:</th>
</tr>
</thead>
</table>

• 2: Enables high availability if the schema design meets requirements for K-safety=2
• 1: Enables high availability if the schema design meets requirements for K-safety=1
• 0: Disables high availability

Return Messages
If you specify a k value of 1 or 2, Vertica returns one of the following messages.

Success:
Marked design n-safe

Failure:
The schema does not meet requirements for K=n.
Fact table projection projection-name
has insufficient “buddy” projections.
where n is a K-safety setting.

Privileges
Superuser

Notes
• The database's internal recovery state persists across database restarts but it is not checked at startup time.
• When one node fails on a system marked K-safe=1, the remaining nodes are available for DML operations.

Examples
=> SELECT MARK_DESIGN_KSAFE(1);
   mark_design_ksafe
----------------------
   Marked design 1-safe
   (1 row)
If the physical schema design is not K-safe, messages indicate which projections do not have a buddy:

```sql
=> SELECT MARK DESIGN KSAFE(1);
The given K value is not correct;
the schema is 0-safe
Projection pp1 has 0 buddies,
which is smaller that the given K of 1
Projection pp2 has 0 buddies,
which is smaller that the given K of 1
.
.
.
(1 row)
```

See Also

- Identical Segmentation
- Failure Recovery

CFS Security Functions

This section describes functions you can use with the Connector Framework Services (CFS) security features.

These functions reside in the v__idol schema in the idollib library that is installed with CFS. When you run one of these functions, you must qualify its name with the v__idol schema. For example:

```sql
=> SELECT v__idol.DELETE_COMMUNITY_KEY();
```

DELETE_COMMUNITY_KEY

Removes the security key from the Vertica database DFS.

Syntax

```sql
V__IDOL.DELETE_COMMUNITY_KEY()
```

Privileges

Superuser
Examples

```sql
=> SELECT V_IDOL.DELETE_COMMUNITY_KEY();
DELETE_COMMUNITY_KEY
-----------------------------
key successfully deleted
```

**DESCRIBE_COMMUNITY_KEY**

Retrieves the value of the security key so you can use it in a query.

**Syntax**

```sql
V_IDOL.DESCRIBE_COMMUNITY_KEY()
```

**Example**

This example retrieves the security key stored in the the Vertica database DFS:

```sql
=> SELECT V_IDOL.DESCRIBE_COMMUNITY_KEY();
describe_community_key
-----------------------------
123.144,564,231
(1 row)
```

**IDOL_CHECK_ACL**

Checks the access control list to verify that the user has permissions to access the data in the IDOL flex table and any views created from the table.

**Syntax**

```sql
v_idol.idol_check_acl(autonomymetadata,securitysection, securitytype using parameters
sis='MTMyfNudFzPXVB5rX2dLnU89Cqo+8668H6iNjHr1l4KPC+ptoA4x8OKzefPMe/y3QNYzQs+QVMd7nmvPc4CaT9qxNr3t2EMgr8YN
GJ5KknE61YN4BixkKd241tEN1lZ08c
CJXiQaTppIqay08UI9aC+JVvtGxeycO03cqrvQiMqxmzmrHEQfw==') from v_idol.idolsecurity;
```

You can also use the session parameter as part of the syntax:

```sql
Alter session set UDPARAMETER FOR v_idol.IdolLib IdolSecurityInfo ='MTMyf ... <encrypted SIS> ...BCmky';
```
SELECT v_idol.idol_check_acl(acl, security_section, security_type) from v_idol.t;

Privileges

To run idol_check_acl on a view, you must be granted access to the view by the database administration user. This example creates a table, creates a view from the table, and grants access to the view to user1:

=> CREATE TABLE idoldata_base(...);
=> CREATE VIEW idoldata as select * from idoldata_base where idol_check_acl(...);
=> GRANT SELECT ON idoldata to user1;

Returns

This function returns one of the following:

- t - indicates the user has permission to access the data.
- f - indicates the user does not have permission to access the data.

INSTALL_COMMUNITY_KEY

Stores the Connector Framework Service security key in the Vertica Distributed File System (DFS).

Syntax

V_IDOL.INSTALL_COMMUNITY_KEY(USING PARAMETERS file_path='keyfile-path');

Parameters

| file_path | Specifies the location of key file from which the key is copied to DFS. |

Privileges

You must be a dbadmin user.
Example

Store the CFS security key that is in btea.key:

```sql
=> SELECT v_idol.install_community_key(
    USING PARAMETERS file_path='/home/release/IDOL/test-data/btea.key');
```

Client Connection Management Functions

This section contains client connection management functions specific to Vertica.

**GET_CLIENT_LABEL**

Returns the client connection label for the current session.

**Syntax**

```sql
GET_CLIENT_LABEL()
```

**Privileges**

None

**Examples**

Return the current client connection label:

```sql
=> SELECT GET_CLIENT_LABEL();
   GET_CLIENT_LABEL
-----------------------
   data_load_application
(1 row)
```

**See Also**

[Setting a Client Connection Label](#)
**RESET_LOAD_BALANCE_POLICY**

Resets the counter each host in the cluster maintains, to track which host it will refer a client to when the native connection load balancing scheme is set to ROUNDROBIN. To reset the counter, run this function on all cluster nodes.

**Syntax**

RESET_LOAD_BALANCE_POLICY()

**Privileges**

Superuser

**Example**

```sql
=> SELECT RESET_LOAD_BALANCE_POLICY();

RESET_LOAD_BALANCE_POLICY

Successfully reset stateful client load balance policies: "roundrobin".
(1 row)
```

**SET_CLIENT_LABEL**

Assigns a label to a client connection for the current session. You can use this label to distinguish client connections.

Labels appear in the v_monitor.sessions table. However, they are not updated in Data Collector tables because the change occurs after Vertica makes the connection.

**Syntax**

SET_CLIENT_LABEL('Label-name')

**Parameters**

| Label-name | VARCHAR name assigned to the client connection label. |
Privileges
None

Examples
Assign label data_load_application to the current client connection:

```sql
=> SELECT SET_CLIENT_LABEL('data_load_application');
SET_CLIENT_LABEL
-----------------------
client_label set to data_load_application
(1 row)
```

See Also
Setting a Client Connection Label

SET_LOAD_BALANCE_POLICY

Sets how native connection load balancing chooses a host to handle a client connection.

Syntax

```
SET_LOAD_BALANCE_POLICY('policy')
```

Parameters

<table>
<thead>
<tr>
<th>policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>The name of the load balancing policy to use, one of the following:</td>
</tr>
<tr>
<td></td>
<td>- NONE (default): Disables native connection load balancing.</td>
</tr>
<tr>
<td></td>
<td>- ROUNDROBIN: Chooses the next host from a circular list of hosts in the cluster that are up—for example, in a three-node cluster, iterates over node1, node2, and node3, then wraps back to node1. Each host in the cluster maintains its own pointer to the next host in the circular list, rather than there being a single cluster-wide state.</td>
</tr>
<tr>
<td></td>
<td>- RANDOM: Randomly chooses a host from among all hosts in the cluster that are up.</td>
</tr>
</tbody>
</table>
Note: Even if the load balancing policy is set on the server to something other than NONE, clients must indicate they want their connections to be load balanced by setting a connection property.

Privileges
Superuser

Example

The following example demonstrates enabling native connection load balancing on the server by setting the load balancing scheme to ROUNDROBIN:

```sql
=> SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');
SET_LOAD_BALANCE_POLICY
--------------------------------------------------------------
Successfully changed the client initiator load balancing policy to: roundrobin
(1 row)
```

See Also

About Native Connection Load Balancing

Cluster Management Functions

This section contains functions that manage spread deployment on large, distributed database clusters.

**REALIGN_CONTROL_NODES**

Chooses control nodes (spread hosts) from all cluster nodes and assigns the rest of the nodes in the cluster to a control node. Calling this function respects existing fault groups, which you can view by querying the `V_CATALOG.CLUSTER_LAYOUT` system table. This view also lets you see the proposed new layout for nodes in the cluster.

**Syntax**

```sql
REALIGN_CONTROL_NODES()
```
Privileges
Superuser

Example
Choose control nodes from all cluster nodes and assign the remaining nodes to a control node:

```sql
=> SELECT REALIGN_CONTROL_NODES;
```

See Also
Defining and Realigning Control Nodes on an Existing Cluster

REBALANCE_CLUSTER

Rebalances the database cluster synchronously as a session foreground task. REBALANCE_CLUSTER returns only after the rebalance operation is complete. If the current session ends, the operation immediately aborts. To rebalance the cluster as a background task, call START_REBALANCE_CLUSTER.

On large cluster arrangements, you typically call REBALANCE_CLUSTER in a flow (see Defining and Realigning Control Nodes in the Administrator's Guide). After you change the number and distribution of control nodes (spread hosts), you must run REBALANCE_CLUSTER() for fault tolerance to be realized.

For detailed information about rebalancing tasks, see Rebalancing Data Across Nodes.

Tip: By default, before performing a rebalance, Vertica queries system tables to compute the size of all projections involved in the rebalance task. This query can add significant overhead to the rebalance operation. To disable this query, set projection configuration parameter `RebalanceQueryStorageContainers` to 0.

Syntax

REBALANCE_CLUSTER()

Privileges
Superuser
Example

```sql
=> SELECT REBALANCE_CLUSTER();
REBALANCE_CLUSTER
-----------------------
REBALANCED
(1 row)
```

**RELOAD_SPREAD**

Updates cluster changes to the catalog's spread configuration file. These changes include:

- New or realigned control nodes
- New spread hosts or fault group
- New or dropped cluster nodes

This function is often used in a multi-step process for large and elastic cluster arrangements. Calling it might require you to restart the database. You must then rebalance the cluster to realize fault tolerance. For details, see Defining and Realigning Control Nodes in the Administrator's Guide.

**Syntax**

`RELOAD_SPREAD( true )`

**Parameters**

<table>
<thead>
<tr>
<th><code>true</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates cluster changes related to control message responsibilities to the spread configuration file.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Example**

Update the cluster with changes to control messaging:
=> SELECT reload_spread(true);
reload_spread
-------------
reloaded
(1 row)

See Also

REBALANCE_CLUSTER

SET_CONTROL_SET_SIZE

Specifies the number of cluster nodes on which to deploy control messaging (spread). You can call this function only on a running database.

To determine whether the current spread hosts and the control designations in the catalog match, query system table LARGE_CLUSTER_CONFIGURATION_STATUS

Note: This function is equivalent to:

install_vertica --large cluster integer

For details, see Installing a Large Cluster in the Administrator's Guide.

Syntax

SET_CONTROL_SET_SIZE(integer)

Parameters

| integer   | The number of cluster hosts from the database cluster on which spread runs. |

Privileges

Superuser

Example

Run spread on two cluster nodes:
See Also

Defining and Realigning Control Nodes on an Existing Cluster
Cluster Scaling Functions

This section contains functions that control how the cluster organizes data for rebalancing.

CANCEL_REBALANCE_CLUSTER

Stops any rebalance task that is currently in progress or is waiting to execute.

Syntax

CANCEL_REBALANCE_CLUSTER()

Privileges

Superuser

Example

```sql
=> SELECT CANCEL_REBALANCE_CLUSTER();
CANCEL_REBALANCE_CLUSTER
------------------------
CANCELED
(1 row)
```

See Also

- START_REBALANCE_CLUSTER
- REBALANCE_CLUSTER

DISABLE_ELASTIC_CLUSTER

Disables elastic cluster scaling, which prevents Vertica from bundling data into chunks that are easily transportable to other nodes when performing cluster resizing. The main reason to disable elastic clustering is if you find that the slightly unequal data distribution in your cluster caused by grouping data into discrete blocks results in performance issues.
Syntax

DISABLE_ELASTIC_CLUSTER()

Privileges

Superuser

Example

```sql
=> SELECT DISABLE_ELASTIC_CLUSTER();
DISABLE_ELASTIC_CLUSTER
------------------------
DISABLED
(1 row)
```

See Also

- [ENABLE_ELASTIC_CLUSTER](#)

**DISABLE_LOCAL_SEGMENTS**

Disables local data segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See [Local Data Segmentation](#) in the Administrator's Guide for details.

Syntax

DISABLE_LOCAL_SEGMENTS()

Privileges

Superuser

Example

```sql
=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
------------------------
DISABLED
```
ENABLE_ELASTIC_CLUSTER

Enables elastic cluster scaling, which makes enlarging or reducing the size of your database cluster more efficient by segmenting a node's data into chunks that can be easily moved to other hosts.

Syntax

ENABLE_ELASTIC_CLUSTER()

Privileges

Superuser

Example

=> SELECT ENABLE_ELASTIC_CLUSTER();
 ENABLE_ELASTIC_CLUSTER
 ------------------------
 ENABLED
 (1 row)

See Also

- DISABLE_ELASTIC_CLUSTER

ENABLE_LOCAL_SEGMENTS

Enables local storage segmentation, which breaks projections segments on nodes into containers that can be easily moved to other nodes. See Local Data Segmentation in the Administrator's Guide for more information.

Syntax

ENABLE_LOCAL_SEGMENTS()
Privileges
Superuser

Example

```sql
=> SELECT ENABLE_LOCAL_SEGMENTS();
   ENABLE_LOCAL_SEGMENTS
--------------------------
   ENABLED
   (1 row)
```

SET_SCALING_FACTOR

Sets the scaling factor that determines the number of storage containers used when rebalancing the database and when using local data segmentation is enabled. See Cluster Scaling for details.

Syntax

```
SET_SCALING_FACTOR( factor )
```

Parameters

| factor | An integer value between 1 and 32. Vertica uses this value to calculate the number of storage containers each projection is broken into when rebalancing or when local data segmentation is enabled. |

Privileges
Superuser

Best Practices

The scaling factor determines the number of storage containers that Vertica uses to store each projection across the database during rebalancing when local segmentation is enabled. When setting the scaling factor, follow these guidelines:
The number of storage containers should be greater than or equal to the number of partitions multiplied by the number of local segments:

\[ \text{num-storage-containers} \geq (\text{num-partitions} \times \text{num-local-segments}) \]

Set the scaling factor high enough so rebalance can transfer local segments to satisfy the skew threshold, but small enough so the number of storage containers does not result in too many ROS containers, and cause ROS pushback. The maximum number of ROS containers is 1024.

Example

```sql
=> SELECT SET_SCALING_FACTOR(12);
SET_SCALING_FACTOR
------------------------
SET
(1 row)
```

**START_REBALANCE_CLUSTER**

Asynchronously rebalances the database cluster as a background task. This function returns immediately after the rebalancing operation is complete. Rebalancing persists until the operation is complete, even if you close the current session or the database shuts down. In the case of shutdown, rebalancing resumes after the cluster restarts. To stop the rebalance operation, call `CANCEL_REBALANCE_CLUSTER`.

For detailed information about rebalancing tasks, see [Rebalancing Data Across Nodes](#).

Syntax

```
START_REBALANCE_CLUSTER()
```

Privileges

Superuser

Example

```sql
=> SELECT START_REBALANCE_CLUSTER();
START_REBALANCE_CLUSTER
--------------------------
```
See Also

REBALANCE_CLUSTER

Communications Functions

This section contains communication functions specific to Vertica.

NOTIFY

Specifies the text message to include with a notification.

Syntax

```
NOTIFY ('message', 'notifier', 'target-topic')
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to send to the end point.</td>
</tr>
<tr>
<td>notifier</td>
<td>The name of the notifier used to deliver the message.</td>
</tr>
<tr>
<td>target-topic</td>
<td>The name of the destination Kafka topic for the message.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

Examples

Send a message to confirm that an ETL job is complete:

```
=> SELECT NOTIFY('ETL Done!', 'my_notifier', 'DB_activity_topic');
```
Constraint Management Functions

This section contains constraint management functions specific to Vertica.
See also SQL system table \texttt{V\_CATALOG\_TABLE\_CONSTRAINTS}.

\textbf{ANALYZE\_CONSTRAINTS}

Analyzes and reports on constraint violations within the specified scope

You can enable automatic enforcement of primary key, unique key, and check constraints when \texttt{INSERT}, \texttt{UPDATE}, \texttt{MERGE}, or \texttt{COPY} statements execute. Alternatively, you can use \texttt{ANALYZE\_CONSTRAINTS} to validate constraints after issuing these statements. Refer to \texttt{Enforcing Primary Key, Unique Key, and Check Constraints Automatically} for more information.

\texttt{ANALYZE\_CONSTRAINTS} performs a lock in the same way that \texttt{SELECT * FROM \textit{table}} holds a lock on \textit{table}. See \texttt{LOCKS} for additional information.

\textbf{Syntax}

\texttt{ANALYZE\_CONSTRAINTS (\'[[database.]schema.]table \'} [, \texttt{column},...]\texttt{\})}

\textbf{Parameters}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{database}</td>
<td>Specifies a schema, by default public. If \textit{schema} is any schema other than public, you must supply the schema name. For example: \texttt{myschema.thisDbObject} If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>\textit{table}</td>
<td>Identifies the table to analyze. If you omit specifying a schema, Vertica uses the current schema search path. If set to an empty string, Vertica analyzes all tables in the current schema.</td>
</tr>
<tr>
<td>\textit{column}</td>
<td>The column in \textit{table} to analyze. You can specify multiple comma-delimited columns. Vertica narrows the scope of the analysis to the specified columns. If you omit specifying a column, Vertica analyzes all columns in \textit{table}.</td>
</tr>
</tbody>
</table>
Privileges

- SELECT privilege on table
- USAGE privilege on schema

Detecting Constraint Violations During a Load Process

Vertica checks for constraint violations when queries are run, not when data is loaded. To detect constraint violations as part of the load process, use a `COPY` statement with the `NO COMMIT` option. By loading data without committing it, you can run a post-load check of your data using the `ANALYZE_CONSTRAINTS` function. If the function finds constraint violations, you can roll back the load because you have not committed it.

If `ANALYZE_CONSTRAINTS` finds violations, such as when you insert a duplicate value into a primary key, you can correct errors using the following functions. Effects last until the end of the session only:

- `DISABLE_DUPLICATE_KEY_ERROR`
- `REENABLE_DUPLICATE_KEY_ERROR`

Important: If a check constraint SQL expression evaluates to an unknown for a given row because a column within the expression contains a null, the row passes the constraint condition.

Return Values

`ANALYZE_CONSTRAINTS` returns results in a structured set (see table below) that lists the schema name, table name, column name, constraint name, constraint type, and the column values that caused the violation.

If the result set is empty, then no constraint violations exist; for example:

```
> SELECT ANALYZE_CONSTRAINTS ('public.product_dimension', 'product_key');
```

```
<table>
<thead>
<tr>
<th>Schema Name</th>
<th>Table Name</th>
<th>Column Names</th>
<th>Constraint Name</th>
<th>Constraint Type</th>
<th>Column Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(0 rows)
```

The following result set shows a primary key violation, along with the value that caused the violation (`'10'`):

```
<table>
<thead>
<tr>
<th>Schema Name</th>
<th>Table Name</th>
<th>Column Names</th>
<th>Constraint Name</th>
<th>Constraint Type</th>
<th>Column Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>product_dimension</td>
<td>product_key</td>
<td>PRIMARY KEY</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
(1 row)
```
The result set columns are described in further detail in the following table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Name</td>
<td>VARCHAR</td>
<td>The name of the schema.</td>
</tr>
<tr>
<td>Table Name</td>
<td>VARCHAR</td>
<td>The name of the table, if specified.</td>
</tr>
<tr>
<td>Column Names</td>
<td>VARCHAR</td>
<td>A list of comma-delimited columns that contain constraints.</td>
</tr>
<tr>
<td>Constraint Name</td>
<td>VARCHAR</td>
<td>The given name of the primary key, foreign key, unique, check, or not null constraint, if specified.</td>
</tr>
<tr>
<td>Constraint Type</td>
<td>VARCHAR</td>
<td>Identified by one of the following strings:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PRIMARY KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FOREIGN KEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CHECK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NOT NULL</td>
</tr>
<tr>
<td>Column Values</td>
<td>VARCHAR</td>
<td>Value of the constraint column, in the same order in which Column Names contains the value of that column in the violating row. When interpreted as SQL, the value of this column forms a list of values of the same type as the columns in Column Names; for example: ('1'), ('1', 'z')</td>
</tr>
</tbody>
</table>

Examples

See Detecting Constraint Violations with ANALYZE_CONSTRAINTS in the Administrator's Guide.
ANALYZE_CORRELATIONS

Analyzes the specified tables for pairs of columns that are strongly correlated. ANALYZE_CORRELATIONS stores the 20 pairs with the strongest correlation. ANALYZE_CORRELATIONS also analyzes statistics.

ANALYZE_CORRELATIONS analyzes only pairwise single-column correlations.

For example, state name and country name columns are strongly correlated because the city name usually, but perhaps not always, identifies the state name. The city of Conshohoken is uniquely associated with Pennsylvania, while the city of Boston exists in Georgia, Indiana, Kentucky, New York, Virginia, and Massachusetts. In this case, city name is strongly correlated with state name.

Behavior Type

Immutable

Syntax

ANALYZE_CORRELATIONS ( '[[[database.]schema.]table ]', [ 'recalculate' ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:  

myschema.thisDbObject  

If you specify a database, it must be the current database. |
| table-name | Identifies the table to analyze. If you omit specifying a schema, Vertica uses the current schema search path. If set to an empty string, Vertica analyzes all tables in the current schema. |
| recalculate | Boolean that specifies whether to analyze correlated columns that were previously analyzed.  

**Note**: Column correlation analysis typically needs to be done only once. |
| Default: false | |

Vertica Documentation  
SQL Reference Manual
Privileges

One of the following:

- Superuser
- User with USAGE privilege on the design schema

Analysis Follow-Up

To take advantage of the correlations that ANALYZE_CORRELATIONS discovers, run Database Designer programmatically. Run DESIGNER_SET_ANALYZE_CORRELATIONS_MODE to specify that Database Designer consider existing column correlations. Be sure to specify that Database Designer not analyze statistics so it does not override existing statistics.

Example

In the following example, ANALYZE_CORRELATIONS analyzes column correlations for all tables in the public schema, even if they currently exist. The correlations that ANALYZE_CORRELATIONS finds are saved, so Database Designer can use them the next time it runs on the VMart database:

```sql
=> SELECT ANALYZE_CORRELATIONS ('public.*', 'true');
ANALYZE_CORRELATIONS
--------------------
0
(1 row)
```

**DISABLE_DUPLICATE_KEY_ERROR**

Disables error messaging when Vertica finds duplicate primary or unique key values at run time (for use with key constraints that are not automatically enabled). Queries execute as though no constraints are defined on the schema. Effects are session scoped.

Syntax

```
DISABLE_DUPLICATE_KEY_ERROR();
```

Privileges

Superuser
Examples

When you call DISABLE_DUPLICATE_KEY_ERROR, Vertica issues warnings letting you know that duplicate values will be ignored, and incorrect results are possible. DISABLE_DUPLICATE_KEY_ERROR is for use only for key constraints that are not automatically enabled.

```sql
=> select DISABLE_DUPLICATE_KEY_ERROR();
WARNING 3152: Duplicate values in columns marked as UNIQUE will now be ignored for the remainder of your session or until reenable_duplicate_key_error() is called
WARNING 3539: Incorrect results are possible. Please contact Vertica Support if unsure
disable_duplicate_key_error
Duplicate key error disabled
(1 row)
```

See Also

ANALYZE_CONSTRAINTS

LAST_INSERT_ID

Returns the last value of an AUTO_INCREMENT/IDENTITY column. If multiple sessions concurrently load the same table with an AUTO_INCREMENT/IDENTITY column, the function returns the last value generated for that column.

Note: This function works only with AUTO_INCREMENT/IDENTITY columns. It does not work with named sequences.

Behavior Type

Volatile

Syntax

LAST_INSERT_ID()

Privileges

- Table owner
- USAGE privileges on the table schema
Examples

See `AUTO_INCREMENT` and `IDENTITY Sequences` in the Administrator's Guide.

**REENABLE_DUPLICATE_KEY_ERROR**

Restores the default behavior of error reporting by reversing the effects of `DISABLE_DUPLICATE_KEY_ERROR`. Effects are session-scoped.

Syntax

```sql
REENABLE_DUPLICATE_KEY_ERROR();
```

Privileges

Superuser

Examples

```sql
=> SELECT REENABLE_DUPLICATE_KEY_ERROR();
REENABLE_DUPLICATE_KEY_ERROR
-----------------------------
Duplicate key error enabled
(1 row)
```

See Also

`ANALYZE_CONSTRAINTS`
Data Collector Functions

The Vertica Data Collector is a utility that extends system table functionality by providing a framework for recording events. It gathers and retains monitoring information about your database cluster and makes that information available in system tables, requiring few configuration parameter tweaks, and having negligible impact on performance.

Collected data is stored on disk in the DataCollector directory under the Vertica /catalog path. You can use the information the Data Collector retains to query the past state of system tables and extract aggregate information, as well as do the following:

- See what actions users have taken
- Locate performance bottlenecks
- Identify potential improvements to Vertica configuration

Data Collector works in conjunction with an advisor tool called Workload Analyzer, which intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history.

By default, Data Collector is on and retains information for all sessions. If performance issues arise, a superuser can disable DC. See Data Collector Parameters and Enabling and Disabling Data Collector in the Administrator's Guide.

This section describes the Data Collection control functions.

Related Topics

V_MONITOR.DATA_COLLECTOR
Retaining Monitoring Information and Analyzing Workloads in the Administrator's Guide

CLEAR_DATA_COLLECTOR

Clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the system table DATA_COLLECTOR. A superuser can clear Data Collector data for all components or specify an individual component

After you clear the Data Collector log, the information is no longer available for querying.
Syntax

CLEAR_DATA_COLLECTOR( [ 'component' ] )

Parameters

| component | Clears memory and disk records for the specified component only. If you provide no argument, the function clears all Data Collector memory and disk records for all components. For the current list of component names, query the system table DATA_COLLECTOR. |

Privileges

Superuser

Examples

The following command clears memory and disk records for the ResourceAcquisitions component:

```sql
=> SELECT clear_data_collector('ResourceAcquisitions');
clear_data_collector
-----------------------
CLEAR
(1 row)
```

The following command clears data collection for all components on all nodes:

```sql
=> SELECT clear_data_collector();
clear_data_collector
-----------------------
CLEAR
(1 row)
```

See Also

- DATA_COLLECTOR
- Retaining Monitoring Information
DATA_COLLECTOR_HELP

Returns online usage instructions about the Data Collector, the DATA_COLLECTOR system table, and the Data Collector control functions.

Syntax

DATA_COLLECTOR_HELP()

Privileges

None

Returns

The DATA_COLLECTOR_HELP() function returns the following information:

```sql
=> SELECT DATA_COLLECTOR_HELP();

Usage Data Collector
The data collector retains history of important system activities.
This data can be used as a reference of what actions have been taken
by users, but it can also be used to locate performance bottlenecks,
or identify potential improvements to the Vertica configuration.
This data is queryable via Vertica system tables.

Access a list of data collector components, and some statistics, by running:
SELECT * FROM v_monitor.data_collector;

The amount of data retained by size and time can be controlled with several functions.
To just set the size amount:
set_data_collector_policy(<component>,
<memory retention (KB)>,
<disk retention (KB)>);

To set both the size and time amounts (the smaller one will dominate):
set_data_collector_policy(<component>,
<memory retention (KB)>,
<disk retention (KB)>,
<interval>);

To set just the time amount:
set_data_collector_time_policy(<component>,
<interval>);

To set the time amount for all tables:
set_data_collector_time_policy(<interval>);
```
The current retention policy for a component can be queried with:

\[\text{get\_data\_collector\_policy(}<\text{component}>);\]

Data on disk is kept in the "DataCollector" directory under the Vertica \catalog path. This directory also contains instructions on how to load the monitoring data into another Vertica database.

To move the data collector logs and instructions to other storage locations, create labeled storage locations using \text{add\_location} and then use:

\[\text{set\_data\_collector\_storage\_location(}<\text{storage\_label}>);\]

Additional commands can be used to configure the data collection logs.

The log can be cleared with:

\[\text{clear\_data\_collector(}[<\text{optional\ component}>]);\]

The log can be synchronized with the disk storage using:

\[\text{flush\_data\_collector(}[<\text{optional\ component}>]);\]

See Also

- DATA_COLLECTOR
- TUNING_RECOMMENDATIONS
- Analyzing Workloads
- Retaining Monitoring Information

**FLUSH_DATA_COLLECTOR**

Waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the log with the disk storage. A superuser can flush Data Collector information for an individual component or for all components.

**Syntax**

\[\text{FLUSH\_DATA\_COLLECTOR(}[\ 'component'\ ] )\]

**Parameters**

| component | Flushes the specified component. If you provide no argument, the function flushes the Data Collector in full. For the current list of component names, query the V_MONITOR.DATA_|
COLLECTOR system table.

Privileges
Superuser

Examples

The following command flushes the Data Collector for the ResourceAcquisitions component:

```sql
=> SELECT flush_data_collector('ResourceAcquisitions');
flush_data_collector
------------------
FLUSH
(1 row)
```

The following command flushes data collection for all components:

```sql
=> SELECT flush_data_collector();
flush_data_collector
------------------
FLUSH
(1 row)
```

See Also

- DATA_COLLECTOR
- Retaining Monitoring Information

GET_DATA_COLLECTOR_POLICY

Retrieves a brief statement about the retention policy for the specified component.

Syntax

```
GET_DATA_COLLECTOR_POLICY( 'component' )
```

Parameters

| component | Returns the retention policy for the specified component. |
For a current list of component names, query the V_MONITOR.DATA_COLLECTOR system table

Privileges

None

Example

The following query returns the history of all resource acquisitions by specifying the ResourceAcquisitions component:

```sql
=> SELECT get_data_collector_policy('ResourceAcquisitions');
get_data_collector_policy
-------------------------------------
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```

See Also

- DATA_COLLECTOR
- Retaining Monitoring Information

SET_DATA_COLLECTOR_POLICY

Sets a size restraint (memory and disk space in kilobytes) for the specified Data Collector table on all nodes. If nodes are down, the failed nodes receive the setting when they rejoin the cluster.

You can use this function to set a size restraint only, or you can include the optional `interval` argument to set disk capacity for both size and time in a single command.

If you specify `interval`, Vertica enforces the setting that is exceeded first (size or time). Before you include a time restraint, verify that the disk size capacity is sufficiently large.

If you want to specify just a time restraint, or you want to turn off a time restraint you set using this function, see `SET_DATA_COLLECTOR_TIME_POLICY()`.

Syntax

```sql
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' [, 'interval'] )
```
Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>component</code></td>
<td>Configures the retention policy for the specified component.</td>
</tr>
<tr>
<td><code>memoryKB</code></td>
<td>Specifies the memory size to retain in kilobytes.</td>
</tr>
<tr>
<td><code>diskKB</code></td>
<td>Specifies the disk size in kilobytes.</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>[Default off] Takes an optional <code>interval</code> argument to specify how long to retain the specified component on disk. To disable a time restraint, set <code>interval</code> to -1.</td>
</tr>
</tbody>
</table>

Note: Any negative input turns off the time restraint

Privileges

Superuser

Notes

- Before you change a retention policy, view its current setting by calling the `GET_DATA_COLLECTOR_POLICY()` function.

- If you don't know the name of a component, query the `V_MONITOR.DATA_COLLECTOR` system table for a list; for example:

  ```
  => SELECT DISTINCT component, description FROM data_collector
  ORDER BY 1 ASC;
  ```

Examples

The following command returns the retention policy for the `ResourceAcquisitions` component:

```
=> SELECT get_data_collector_policy('ResourceAcquisitions');
get_data_collector_policy
----------------------------------------
1000KB kept in memory, 10000KB kept on disk.
(1 row)
```
This command changes the memory and disk setting for ResourceAcquisitions from its current setting of 1,000 KB memory and 10,000 KB disk space to 1500 KB and 25000 KB, respectively:

```sql
=> SELECT set_data_collector_policy('ResourceAcquisitions', '1500', '25000');
set_data_collector_policy
------------------------
SET
(1 row)
```

This command sets the RequestsIssued component to 1500 KB memory and 11000 KB on disk, and includes a 3-minute time restraint:

```sql
=> SELECT set_data_collector_policy('RequestsIssued', '1500', '11000', '3 minutes '::interval);
set_data_collector_policy
------------------------
SET
(1 row)
```

The following command disables the 3-minute retention policy for the RequestsIssued component, using SET_DATA_COLLECTOR_TIME_POLICY:

```sql
=> SELECT set_data_collector_time_policy('RequestsIssued', '-1');
set_data_collector_time_policy
------------------------
SET
(1 row)
```

See Also

- GET_DATA_COLLECTOR_POLICY
- SET_DATA_COLLECTOR_TIME_POLICY()
- DATA_COLLECTOR
- Retaining Monitoring Information in the Administrator's Guide

**SET_DATA_COLLECTOR_TIME_POLICY**

Sets a time capacity for individual Data Collector tables on all nodes. If nodes are down, the failed nodes receive the setting when they rejoin the cluster.

If you specify *interval*, Vertica enforces the setting that is exceeded first (size or time). Before you include a time restraint, verify that the disk size capacity is sufficiently large.

If you want to configure both time and size restraints at the same time, see SET_DATA_COLLECTOR_POLICY.
Syntax

```
SET_DATA_COLLECTOR_TIME_POLICY( ['component'], 'interval' )
```

Parameters

| component | [Optional] Configures the time retention policy for the specified component. If you omit the component argument, Vertica sets the specified time capacity for all Data Collector tables. |
| interval | Specifies the time restraint on disk using an INTERVAL type. To disable a time restraint, set interval to -1. |

Note: Any negative input turns off the time restraint

Privileges

Superuser

Usage Considerations

- Before you change a retention policy, view its current setting by calling the `GET_DATA_COLLECTOR_POLICY` function.

- If you don't know the name of a component, query the V_MONITOR.DATA_COLLECTOR system table for a list. For example:

  ```
  => SELECT DISTINCT component, description FROM data_collector
  ORDER BY 1 ASC;
  ```

Setting the Time Interval for System Tables

You can also use the `interval` argument to query system tables the same way you query Data Collector tables; for example:

```
set_data_collector_time_policy('<system-table>', '<interval>');
```

To illustrate, the following command in the left column is equivalent to running the series of commands on the right:
Run one command | Instead of a series of commands
-----------------|--------------------------------------
SELECT set_data_collector_time_policy ('v_monitor.query_requests', '3 minutes'::interval); | SELECT set_data_collector_time_policy ('RequestsIssued', '3 minutes'::interval);

SELECT set_data_collector_time_policy ('RequestsCompleted', '3 minutes'::interval);

SELECT set_data_collector_time_policy ('RequestsCompleted', '3 minutes'::interval);

SELECT set_data_collector_time_policy ('RequestsCompleted', '3 minutes'::interval);

The **SET_DATA_COLLECTOR_TIME_POLICY** function updates the time capacity for all Data Collector tables in the V_MONITOR.QUERY_REQUESTS view. The new setting overrides any previous settings for every Data Collector table in that view.

Examples

The following command configures the Backups component to be retained on disk for 1 day:

```sql
=> SELECT set_data_collector_time_policy('Backups', '1 day'::interval);
set_data_collector_time_policy
----------------------------------
 SET
(1 row)
```

This command disables the 1-day restraint for the Backups component:

```sql
=> SELECT set_data_collector_time_policy('Backups', '-1');
set_data_collector_time_policy
--------------------------
 SET
(1 row)
```

This command sets a 30-minute time capacity for all Data Collector tables in a single command:

```sql
=> SELECT set_data_collector_time_policy('30 minutes'::interval);
set_data_collector_time_policy
--------------------------
 SET
(1 row)
```

To view current retention policy settings for each Data Collector table, call the **GET_DATA_COLLECTION_POLICY( )** function. In the next example, the time restraint is included.
=> SELECT get_data_collector_policy('RequestsIssued');
                   get_data_collector_policy
-----------------------------------------------
2000KB kept in memory, 50000KB kept on disk. 2 years 3 days 15:08 hours kept on disk.
(1 row)

If the time policy setting is disabled, the output of GET_DATA_COLLECTION_POLICY() returns "Time based retention disabled."

2000KB kept in memory, 50000KB kept on disk. Time based retention disabled.

See Also

- GET_DATA_COLLECTOR_POLICY
- SET_DATA_COLLECTOR_POLICY
- DATA_COLLECTOR
Database Designer Functions

Database Designer functions perform the following operations, generally performed in the following order:

1. **Create a design.**
2. **Set design properties.**
3. **Populate a design.**
4. **Create design and deployment scripts.**
5. **Get design data.**
6. **Clean up.**

For detailed information, see *Workflow for Running Database Designer Programmatically*. For information on required privileges, see *Privileges for Running Database Designer Functions* in the Administrator's Guide.

Create a design

**DESIGNER_CREATE дизайн** directs Database Designer to create a design.

Set design properties

The following functions let you specify properties of a particular design:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_SET_DESIGN_TYPE</td>
<td>Specifies whether the design is comprehensive or incremental.</td>
</tr>
<tr>
<td>DESIGNER_DESIGN_PROJECTION_ENCODINGS</td>
<td>Analyzes encoding in the specified projections and creates a script that implements encoding recommendations.</td>
</tr>
<tr>
<td>DESIGNER_SET_DESIGN_KSAFETY</td>
<td>Sets the K-safety value for a comprehensive design.</td>
</tr>
<tr>
<td>DESIGNER_SET_OPTIMIZATION_OBJECTIVE</td>
<td>Specifies whether the design optimizes for query or load performance.</td>
</tr>
</tbody>
</table>
**DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS**

Enables inclusion of unsegmented projections in the design.

**DESIGNER_SET_ANALYZE_CORRELATIONS_MODE**

Determines how the design handles column correlations.

---

## Populate a design

The following functions let you add tables and queries to your Database Designer design:

<table>
<thead>
<tr>
<th><strong>FUNCTION</strong></th>
<th><strong>DESCRIPTION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESIGNER_ADDDESIGN_TABLES</strong></td>
<td>Adds the specified tables to a design.</td>
</tr>
<tr>
<td><strong>DESIGNER_ADDDESIGN_QUERY</strong></td>
<td>Adds queries to the design and weights them.</td>
</tr>
<tr>
<td><strong>DESIGNER_ADDDESIGN_QUERIES</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DESIGNER_ADDDESIGN_QUERIES_FROMRESULTS</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

## Create design and deployment scripts

The following functions populate the Database Designer workspace and create design and deployment scripts. You can also analyze statistics, deploy the design automatically, and drop the workspace after the deployment:

<table>
<thead>
<tr>
<th><strong>FUNCTION</strong></th>
<th><strong>DESCRIPTION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESIGNER_RUN_POPULATEDESIGN_AND_DEPLOY</strong></td>
<td>Populates the design and creates design and deployment scripts.</td>
</tr>
<tr>
<td><strong>DESIGNER_WAITFORDESIGN</strong></td>
<td>Waits for a currently running design to complete.</td>
</tr>
</tbody>
</table>

---

## Reset a design

**DESIGNER_RESETDESIGN** discards all the run-specific information of the previous Database Designer build or deployment of the specified design but retains its configuration.

## Get design data

The following functions display information about projections and scripts that the Database Designer created:
DESIGNER_OUTPUT_ALL Design_PROJECTIONS Sends to standard output DDL statements that define design projections.

DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT Sends to standard output a design's deployment script.

Clean up

The following functions cancel any running Database Designer operation or drop a Database Designer design and all its contents:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER_CANCEL_POPULATE_DESIGN</td>
<td>Cancels population or deployment operation for the specified design if it is currently running.</td>
</tr>
<tr>
<td>DESIGNER_DROP_DESIGN</td>
<td>Removes the schema associated with the specified design and all its contents.</td>
</tr>
<tr>
<td>DESIGNER_DROP_ALL_DESIGNS</td>
<td>Removes all Database Designer-related schemas associated with the current user.</td>
</tr>
</tbody>
</table>

**DESIGNER_ADD_DESIGN_QUERIES**

Reads and evaluates queries from an input file, and adds the queries that it accepts to the specified design. All accepted queries are assigned a weight of 1 (see DESIGNER_ADD_DESIGN_QUERIES).

The following requirements apply:

- All queried tables must previously be added to the design with DESIGNER_ADD_DESIGN_TABLES.
- If the design type is incremental, the Database Designer reads only the first 100 queries in the input file, and ignores all queries beyond that number.

All accepted queries are added to the system table DESIGN_QUERIES.

Behavior Type

Immutable
Syntax

DESIGNER_ADDDESIGN_QUERIES ( 'design-name', 'input-file' [, 'return-results'] )

Parameters

<table>
<thead>
<tr>
<th>design-name</th>
<th>Name of the target design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-file</td>
<td>Absolute path to the queries file.</td>
</tr>
<tr>
<td>return-results</td>
<td>Boolean, optionally specifies whether to return results of the add operation to standard output. If set to true, Database Designer returns the following results:</td>
</tr>
</tbody>
</table>

- Number of accepted queries
- Number of queries referencing non-design tables
- Number of unsupported queries
- Number of illegal queries

Privileges

Read access to the input file, and one of the following:

- Superuser
- Design creator with all privileges that pertain to the input file queries (see Privileges Required for Common Database Operations).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
  - External or system tables only
  - Local temporary or other non-design tables
DELETE or UPDATE query has one or more subqueries.

INSERT query does not include a SELECT clause.

Database Designer cannot optimize the query.

Examples

The following example adds queries from vmart_queries.sql to the VMART_DESIGN design. This file contains nine queries. The statement includes a third argument of true, so Database Designer returns results of the add operation:

```sql
=> SELECT DESIGNER_ADD DESIGN QUERIES ('VMART_DESIGN', '/tmp/examples/vmart_queries.sql', 'true');
...
DESIGNER_ADD DESIGN QUERIES
---------------------------------------------
Number of accepted queries = 9
Number of queries referencing non-design tables = 0
Number of unsupported queries = 0
Number of illegal queries = 0
(1 row)
```

See Also

Running Database Designer Programmatically

**DESIGNER_ADD DESIGN QUERIES_FROM_RESULTS**

Executes the specified query and evaluates results in the following columns:

- **QUERY_TEXT** (required): Text of potential design queries.

- **QUERY_WEIGHT** (optional): The weight assigned to each query that indicates its importance relative to other queries, a real number >0 and ≤ 1. Database Designer uses this setting when creating the design to prioritize the query. If DESIGNER_ADD DESIGN QUERIES_FROM_RESULTS returns any results that omit this value, Database Designer sets their weight to 1.

After evaluating the queries in QUERY_TEXT, DESIGNER_ADD DESIGN QUERIES_FROM_RESULTS adds all accepted queries to the design. An unlimited number of queries can be added to the design.

Before you add queries to a design, you must add the queried tables with DESIGNER_ADD DESIGN TABLES.
Behavior Type
Immutable

Syntax

DESIGNER_ADD_DESIGN QUERIES_FROM_RESULTS ( 'design-name', 'query' )

Parameters

<table>
<thead>
<tr>
<th>design-name</th>
<th>Name of the target design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>A valid SQL query whose results contain columns named QUERY_TEXT and, optionally, QUERY_WEIGHT.</td>
</tr>
</tbody>
</table>

Privileges

- Superuser
- Design creator with all privileges that pertain to the specified query and all queries that this function returns (see Privileges Required for Common Database Operations).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
  - External or system tables only
  - Local temporary or other non-design tables
- DELETE or UPDATE query has one or more subqueries.
- INSERT query does not include a SELECT clause.
- Database Designer cannot optimize the query.
Example

The following example queries the system table `QUERY_REQUESTS` for all long-running queries (> 1 million microseconds) and adds them to the VMART_DESIGN design. The query returns no information on query weights, so all queries are assigned a weight of 1:

```sql
=> SELECT DESIGNER_ADD_DESIGN_QUERIES_FROM_RESULTS ('VMART_DESIGN',
   'SELECT request as query_text FROM query_requests where request_duration_ms > 1000000 AND request_type = "QUERY";');
```

See Also

Running Database Designer Programmatically

**DESIGNER_ADD_DESIGN_QUERY**

Reads and parses the specified query, and if accepted, adds it to the design. Before you add queries to a design, you must add the queried tables with `DESIGNER_ADD_DESIGN_TABLES`. All accepted queries are added to the system table `DESIGN_QUERIES`.

**Behavior Type**

Immutable

**Syntax**

`DESIGNER_ADD_DESIGN_QUERY ( 'design-name', 'design-query' [, query-weight] )`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>design-name</code></td>
<td>Name of the target design.</td>
</tr>
<tr>
<td><code>design-query</code></td>
<td>Executable SQL query.</td>
</tr>
<tr>
<td><code>query-weight</code></td>
<td>Optionally assigns a weight to each query that indicates its importance</td>
</tr>
<tr>
<td></td>
<td>relative to other queries, a real number &gt;0 and ≤ 1. Database Designer</td>
</tr>
<tr>
<td></td>
<td>uses this setting to prioritize queries in the design.</td>
</tr>
<tr>
<td></td>
<td>If you omit this parameter, Database Designer assigns a weight of 1.</td>
</tr>
</tbody>
</table>
Privileges

One of the following:

- Superuser
- Design creator with all privileges that pertain to the specified query (see Privileges Required for Common Database Operations).

Errors

Database Designer returns an error in the following cases:

- The query contains illegal syntax.
- The query references:
  - External or system tables only
  - Local temporary or other non-design tables
- DELETE or UPDATE query has one or more subqueries.
- INSERT query does not include a SELECT clause.
- Database Designer cannot optimize the query.

Examples

The following example adds the specified query to the VMART_DESIGN design and assigns that query a weight of 0.5:

```sql
-> SELECT DESIGNER_ADD DESIGN_QUERY (  
   'VMART_DESIGN',  
   'SELECT customer_name, customer_type FROM customer_dimension ORDER BY customer_name ASC;', 0.5  
);
```

See Also

Running Database Designer Programmatically
DESIGNER_ADD_DESIGN_TABLES

Adds the specified tables to a design. You must run DESIGNER_ADD_DESIGN_TABLES before adding design queries to the design. If no tables are added to the design, Vertica does not accept design queries.

Behavior Type

Immutable

Syntax

DESIGNER_ADD_DESIGN_TABLES ( 'design-name', '[ table-spec[, ...] ]' [, 'analyze-statistics' ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>design-name</td>
<td>Name of the Database Designer design.</td>
</tr>
<tr>
<td>table-spec[, ...]</td>
<td>One or more comma-delimited arguments that specify which tables to add to the design, where each table-spec argument can specify tables as follows:</td>
</tr>
<tr>
<td></td>
<td>• [schema.]table</td>
</tr>
<tr>
<td></td>
<td>Add table to the design.</td>
</tr>
<tr>
<td></td>
<td>• schema.*</td>
</tr>
<tr>
<td></td>
<td>Add all tables in schema.</td>
</tr>
<tr>
<td>analyze-statistics</td>
<td>Boolean that optionally specifies whether to run ANALYZE_STATISTICS after adding the specified tables to the design, by default set to False.</td>
</tr>
<tr>
<td></td>
<td>Accurate statistics help Database Designer optimize compression and query performance. Updating statistics takes time and resources.</td>
</tr>
</tbody>
</table>
Privileges

One of the following:

- Superuser
- DBDUSER who created the design, has USAGE privilege on the design table schema, and owns the design table.

Examples

The following example adds to design VMART_DESIGN all tables from schemas online_sales and store, and analyzes statistics for those tables:

```sql
=> SELECT DESIGNER_ADD DESIGN TABLES ('VMART DESIGN', 'online_sales.*', 'store.*', 'true');
DESIGNER_ADD DESIGN TABLES
-----------------------------
  7

(1 row)
```

See Also

Running Database Designer Programmatically

**DESIGNERCANCEL_POPULATEDESIGN**

Cancels population or deployment operation for the specified design if it is currently running. When you cancel a deployment, the Database Designer cancels the projection refresh operation. It does not roll back projections that it already deployed and refreshed.

Behavior Type

Immutable

Syntax

```
DESIGNERCANCEL_POPULATEDESIGN ( 'design-name' )
```

Parameters

| design-name | Name of the design operation to cancel. |
Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example cancels a currently running design for VMART_DESIGN and then drops the design:

```sql
=> SELECT DESIGNER_CANCEL_POPULATE_DESIGN ('VMART_DESIGN');
=> SELECT DESIGNER_DROP_DESIGN ('VMART_DESIGN', 'true');
```

See Also

Running Database Designer Programmatically

**DESIGNER_CREATE_DESIGN**

Creates a design with the specified name.

Note: Be sure to back up the current design using the function `EXPORT_CATALOG` before running the Database Designer functions on an existing schema. You must explicitly back up the existing design when using Database Designer programmatically.

If any of the following V_MONITOR tables do not already exist from previous designs, DESIGNER_CREATE_DESIGN creates them:

- **DESIGNS**
- **DESIGN_TABLES**
- **DEPLOYMENT_PROJECTIONS**
- **DEPLOYMENT_PROJECTION_STATEMENTS**
- **DESIGN QUERIES**
Behavior Type
Immutable

Syntax

DESIGNER_CREATE_DESIGN ( 'design-name' )

Parameters

| design-name | Name of the design to create, can contain only alphanumeric and underscore (_) characters. Two users cannot have designs with the same name at the same time. |

Privileges
One of the following:

- Superuser
- DBDUSER

Examples
The following example creates the design VMART_DESIGN:

```sql
=> SELECT DESIGNER_CREATE_DESIGN('VMART_DESIGN');
DESIGNER_CREATE_DESIGN
-----------------------
0
(1 row)
```

See Also

Running Database Designer Programatically
**DESIGNER_DESIGN_PROJECTION_ENCODINGS**

Analyzes encoding in the specified projections, creates a script to implement encoding recommendations, and optionally deploys the recommendations.

**Behavior Type**

Immutable

**Syntax**

```sql
DESIGNER_DESIGN_PROJECTION_ENCODINGS ( '[ proj-spec[, ... ] ]', 'destination' [, 'deploy'] [, 'reanalyze-encodings'] )
```

**Parameters**

<table>
<thead>
<tr>
<th><code>proj-spec[, ...]</code></th>
<th>One or more comma-delimited projections to add to the design. Each projection can be specified in one of the following ways:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• [[schema.]table.]projection</td>
</tr>
<tr>
<td></td>
<td>Specifies to analyze projection.</td>
</tr>
<tr>
<td></td>
<td>• schema.*</td>
</tr>
<tr>
<td></td>
<td>Specifies to analyze all projections in the named schema.</td>
</tr>
<tr>
<td></td>
<td>• [schema.]table</td>
</tr>
<tr>
<td></td>
<td>Specifies to analyze all projections of the named table.</td>
</tr>
<tr>
<td></td>
<td>If set to an empty string, Vertica analyzes all projections in the database to which the user has access.</td>
</tr>
<tr>
<td></td>
<td>For example, the following statement specifies to analyze all projections in schema private, and send the results to the file encodings.sql:</td>
</tr>
<tr>
<td></td>
<td>=&gt; SELECT DESIGNER_DESIGN_PROJECTION_ENCODINGS ('mydb.private.*','encodings.sql');</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>destination</code></th>
<th>Specifies where to send output, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• An empty string ('') writes the script to standard output.</td>
</tr>
</tbody>
</table>
The path and name of a SQL output file. If you specify a file that does not exist, the function creates one. If you specify only a file name, Vertica creates it in the catalog directory. If the file already exists, the function silently overwrites its contents.

| **deploy** | Boolean that optionally specifies whether to deploy encoding changes, by default set to false. |
| **reanalyze-encodings** | Boolean that optionally specifies whether DESIGNER_DESIGN_PROJECTION_ENCODINGS analyzes encodings in a projection where all columns are already encoded: 
- false (default): Analyzes no columns and generates no recommendations if all columns are encoded.
- true: Ignores existing encodings and generates recommendations. |

Privileges

The following requirements pertain to the target projections:

- OWNER of all projections to analyze
- USAGE privilege on the schema for the specified projections

Examples

The following example requests that Database Designer analyze encodings of all projections in the schema online_sales, as follows:

- The second parameter destination is set to an empty string, so the script is sent to standard output (shown truncated below).
- The last two parameters deploy and reanalyze-encodings are omitted, so Database Designer does not execute the script or reanalyze existing encodings:

```sql
=> SELECT DESIGNER_DESIGN_PROJECTION_ENCODINGS ('online_sales.*','');
CREATE PROJECTION online_page_dimension_DBD_1_seg_EncodingDesign /*+Createtype(D)*/
(
  online_page_key ENCODING COMMONDELTA_COMP,
  start_date ENCODING DELTAVAL,
```
end_date ENCODING DELTAVAL,
page_number ENCODING DELTAVAL,
page_description,
page_type
)
AS
SELECT online_page_dimension.online_page_key,
    online_page_dimension.start_date,
    online_page_dimension.end_date,
    online_page_dimension.page_number,
    online_page_dimension.page_description,
    online_page_dimension.page_type
FROM online_sales.online_page_dimension
ORDER BY online_page_dimension.online_page_key
SEGMENTED BY hash(online_page_dimension.online_page_key) ALL NODES KSAFE 1;

select refresh('online_sales.online_page_dimension');
select make_ahm_now();

DROP PROJECTION online_sales.online_page_dimension CASCADE;

ALTER PROJECTION online_sales.online_page_dimension_DBD_1_seg_EncodingDesign RENAME TO online_page_dimension;
...
(1 row)

See Also

Running Database Designer Programmatically

DESIGNER_DROP_ALL_DESIGNS

Removes all Database Designer-related schemas associated with the current user. Use this function to remove database objects after one or more Database Designer sessions complete execution.

Behavior Type

Immutable

Syntax

DESIGNER_DROP_ALL_DESIGNS()
Parameters

None.

Privileges

One of the following:

- Superuser: Drops all designs.
- Design creator: Drops all designs created by this user.

Example

The following example removes all schema and their contents associated with the current user. DESIGNER_DROP_ALL DESIGNS returns the number of designs dropped:

```sql
=> SELECT DESIGNER_DROP_ALL_DESIGNS();
DESIGNER_DROP_ALL_DESIGNS
-----------------------
   2
(1 row)
```

See Also

- DESIGNER_CANCEL_POPULATE_DESIGN
- DESIGNER_DROP_DESIGN

DESIGNER_DROP_DESIGN

Removes the schema associated with the specified design and all its contents. Use DESIGNER_DROP_DESIGN after a Database Designer design or deployment completes successfully. You must also use it to drop a design before creating another one under the same name.

To drop all designs that you created, use DESIGNER_DROP_ALL_DESIGNS.

Behavior Type

Immutable
Syntax

```
DESIGNER_DROP_DESIGN ( 'design-name' [, force-drop ] )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>design-name</code></td>
<td>Name of the design to drop.</td>
</tr>
<tr>
<td><code>force-drop</code></td>
<td>Boolean that overrides any dependencies that otherwise prevent Vertica from executing this function—for example, the design is in use or is currently being deployed. If you omit this parameter, Vertica sets it to false.</td>
</tr>
</tbody>
</table>

Privileges

One of the following:

- Superuser
- Design creator

Example

The following example deletes the Database Designer design VMART_DESIGN and all its contents:

```sql
=> SELECT DESIGNER_DROP_DESIGN ('VMART_DESIGN');
```

See Also

- Running Database Designer Programmatically

**DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS**

Displays the DDL statements that define the design projections to standard output.

Behavior Type

Immutable
Syntax

DESIGNER_OUTPUT_ALL DESIGN PROJECTIONS ( 'design-name' )

Parameters

| design-name | Name of the target design. |

Privileges

One of the following:

- Superuser
- DBDUSER role

Examples

The following example returns the design projection DDL statements for vmart_design:

```sql
=> SELECT DESIGNER_OUTPUT_ALL DESIGN PROJECTIONS('vmart_design');
CREATE PROJECTION customer_dimension_DBD_1_rep_VMART_DESIGN /*+createtype(D)*/
( customer_key ENCODING DELTAVAL,
  customer_type ENCODING AUTO,
  customer_name ENCODING AUTO,
  customer_gender ENCODING REL,
  title ENCODING AUTO,
  household_id ENCODING DELTAVAL,
  customer_address ENCODING AUTO,
  customer_city ENCODING AUTO,
  customer_state ENCODING AUTO,
  customer_region ENCODING AUTO,
  marital_status ENCODING AUTO,
  customer_age ENCODING DELTAVAL,
  number_of_children ENCODING BLOCKDICT_COMP,
  annual_income ENCODING DELTARANGE_COMP,
  occupation ENCODING AUTO,
  largest_bill_amount ENCODING DELTAVAL,
  store_membership_card ENCODING BLOCKDICT_COMP,
  customer_since ENCODING DELTAVAL,
  deal_stage ENCODING AUTO,
  deal_size ENCODING DELTARANGE_COMP,
  last_deal_update ENCODING DELTARANGE_COMP )
AS
SELECT customer_key,
  customer_type,
  customer_name,
```
See Also
DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT

DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT

Displays the deployment script for the specified design to standard output. If the design is already deployed, Vertica ignores this function.

To output only the CREATE PROJECTION commands in a design script, use DESIGNER_OUTPUT_ALL DESIGN PROJECTIONS.

Behavior Type
Immutable

Syntax
DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT ( 'design-name' )
Parameters

| design-name | Name of the target design. |

Privileges

One of the following:

- Superuser
- Design creator

Examples

The following example displays the deployment script for VMART_DESIGN:

```sql
=> SELECT DESIGNER_OUTPUT_DEPLOYMENT_SCRIPT('VMART_DESIGN');
CREATE PROJECTION customer_dimension_DBD_1_rep_VMART_DESIGN /*+createtype(D)/
...
CREATE PROJECTION product_dimension_DBD_2_rep_VMART_DESIGN /*+createtype(D)/
...
select refresh('public.customer_dimension,
    public.product_dimension,
    public.promotion.dimension,
    public.date_dimension');
select make_ahm_now();
DROP PROJECTION public.customer_dimension_super CASCADE;
DROP PROJECTION public.product_dimension_super CASCADE;
...
```

See Also

DESIGNER_OUTPUT_ALL_DESIGN_PROJECTIONS

DESIGNER_RESET_DESIGN

Discards all run-specific information of the previous Database Designer build or deployment of the specified design but keeps its configuration. You can make changes to the design as needed, for example, by changing parameters or adding additional tables and/or queries, before running the design again.

Behavior Type

Immutable
**Syntax**

```
DESIGNER_RESET_DESIGN ( 'design-name' )
```

**Parameters**

| design-name | Name of the design to reset. |

**Privileges**

- Superuser
- Design creator

**Example**

The following example resets the Database Designer design VMART_DESIGN:

```
=> SELECT DESIGNER_RESET_DESIGN ('VMART_DESIGN');
```

**DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY**

Populates the design and creates the design and deployment scripts. DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY can also analyze statistics, deploy the design, and drop the workspace after the deployment.

**Caution:** DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY does not create a backup copy of the current design before deploying the new design. Before running this function, back up the existing schema design with EXPORT_CATALOG.

**Behavior Type**

Immutable

**Syntax**

```
DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY ( 'design-name',
                                             'output-design-file',
                                             'output-deployment-file')
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>design-name</td>
<td>Name of the design to populate and deploy.</td>
</tr>
<tr>
<td>output-design-file</td>
<td>Specifies where to save the file with DDL statements to create design projections, where output-design-file is an absolute path to the node where the session is connected.</td>
</tr>
<tr>
<td>output-deployment-file</td>
<td>Specifies where to save the file that contains the deployment script, where output-deployment-file is an absolute path to the node where the session is connected.</td>
</tr>
<tr>
<td>analyze-statistics</td>
<td>Specifies whether to collect or refresh statistics for the tables before populating the design. If set to true, Vertica Invokes ANALYZE_STATISTICS. Accurate statistics help Database Designer optimize compression and query performance. However, updating statistics requires time and resources. Default: false</td>
</tr>
<tr>
<td>deploy</td>
<td>Specifies whether to deploy the Database Designer design using the deployment script created by this function.                                                                                               Default: true</td>
</tr>
<tr>
<td>drop-design-workspace</td>
<td>Specifies whether to drop the design workspace after the design is deployed.                                                                                                                                  Default: true</td>
</tr>
<tr>
<td>continue-after-error</td>
<td>Specifies whether DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY continues to run after an error occurs. By default, an error causes this function to terminate.                                                     Default: false</td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x)
Privileges

- Superuser
- Design creator with WRITE privileges on storage locations of design and deployment scripts; otherwise DESIGNER_RUN_POPULATEDESIGN_AND_DEPLOY cannot save the design and deployment scripts.

Requirements

Before calling this function, you must:

- Create a design, a logical schema with tables.
- Associate tables with the design.
- Load queries to the design.
- Set design properties (K-safety level, mode, and policy).

Examples

The following example creates projections for and deploys the VMART_DESIGN design, and analyzes statistics about the design tables.

```sql
=> SELECT DESIGNER_RUN_POPULATEDESIGN_AND_DEPLOY (
    'VMART_DESIGN',
    '/tmp/examples/vmart_design_files/vmart_design_DDL',
    '/tmp/examples/vmart_design_files/vmart_design_deployment_scripts',
    'true',
    'false',
    'false',
    'false'
);```

See Also

Running Database Designer Programatically

DESIGNER_SET_ANALYZE_CORRELATIONS_MODE

Specifies how Database Designer handles column correlations in a design. A design's mode determines whether Database Designer analyzes or re-analyzes existing column correlations.
and considers them in the design that it creates.

The following recommendations apply:

- You typically need to analyze column correlations only once.
- Analyze correlations when the table row count is at least \texttt{DBDCorrelationSampleRowCount}—by default, 4000.

\begin{Verbatim}
\textbf{Important:} Database Designer analyzes column correlations for a design only if you enable analysis with this function.
\end{Verbatim}

\textbf{Behavior Type}

Immutable

\textbf{Syntax}

\texttt{DESIGNER\_SET\_ANALYZE\_CORRELATIONS\_MODE ( \texttt{design-name}, mode )}

\textbf{Parameters}

<table>
<thead>
<tr>
<th>\textit{design-name}</th>
<th>Name of the design that specifies how Database Designer handles correlated columns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{mode}</td>
<td>Specifies how to handle correlations in the design tables, one of the following integer settings:</td>
</tr>
<tr>
<td>0</td>
<td>Ignore column correlations in the design tables.</td>
</tr>
<tr>
<td>1</td>
<td>Consider existing correlations in tables when creating the design. If no existing correlations exist, Database Designer sets none in the design.</td>
</tr>
<tr>
<td>2</td>
<td>Analyze column correlations on tables where correlation analysis was not previously performed. When creating the design, consider all column correlations, new and existing.</td>
</tr>
<tr>
<td>3</td>
<td>Analyze all tables for column correlations and consider them when creating the design. If correlations already exist for a table, ignore them and re-analyze the table.</td>
</tr>
</tbody>
</table>
Setting the correlation analysis mode does not affect whether Database Designer analyzes statistics when creating a design.

Privileges

One of the following:

- Superuser
- DBDUSER role with USAGE privilege on the design schema.

Example

The following example specifies that Database Designer analyze all tables for correlated columns and consider them when creating a design:

```sql
=> SELECT DESIGNER_SET_ANALYZE_CORRELATIONS_MODE ('VMARTDESIGN', 3);
DESIGNER_SET_ANALYZE_CORRELATIONS_MODE
-----------------------------
            3
(1 row)
```

See Also

- `ANALYZE_CORRELATIONS`
- `Running Database Designer Programmatically`

**DESIGNER_SET_DESIGN_KSAFETY**

Sets K-safety for a comprehensive design and stores the K-safety value in the `DESIGNS` table. Database Designer ignores this function for incremental designs.

Behavior Type

Immutable

Syntax

```sql
DESIGNER_SET_DESIGN_KSAFETY ( 'design-name' [, k-level ]
)
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>design-name</strong></td>
<td>Name of the design for which you want to set the K-safety value, type VARCHAR.</td>
</tr>
</tbody>
</table>
| **k-level**     | An integer between 0 and 2 that specifies the level of K-safety for the target design. This value must be compatible with the number of nodes in the database cluster:  
- **k-level** = 0: ≥ 1 nodes  
- **k-level** = 1: ≥ 3 nodes  
- **k-level** = 2: ≥ 5 nodes  
If you omit this parameter, Vertica sets K-safety for this design to 0 or 1, according to the number of nodes: 1 if the cluster contains ≥ 3 nodes, otherwise 0.  
If you are a DBADMIN user and **k-level** differs from system K-safety, Vertica changes system K-safety as follows:  
- If **k-level** is less than system K-safety, Vertica changes system K-safety to the lower level after the design is deployed.  
- If **k-level** is greater than system K-safety and is valid for the database cluster, Vertica creates the required number of buddy projections for the tables in this design. If the design applies to all database tables, or all tables in the database have the required number of buddy projections, Database Designer changes system K-safety to **k-level**.  
If the design excludes some database tables and the number of their buddy projections is less than **k-level**, Database Designer leaves system K-safety unchanged. Instead, it returns a warning and indicates which tables need new buddy projections in order to adjust system K-safety.  
If you are a DBDUSER, Vertica ignores this parameter. |

## Privileges

One of the following:
Examples

The following example set K-safety for the VMART_DESIGN design to 1:

```sql
=> SELECT DESIGNER_SET_DESIGN_KSAFETY('VMART_DESIGN', 1);
```

See Also

Running Database Designer Programmatically

**DESIGNER_SET_DESIGN_TYPE**

Specifies whether Database Designer should create a comprehensive or incremental design. DESIGNER_SET_DESIGN_TYPE stores the design mode in the DESIGNS table.

If you do not explicitly set a design mode with this function, Database Designer creates a comprehensive design.

**Behavior Type**

Immutable

**Syntax**

```
DESIGNER_SET_DESIGN_TYPE ( 'design-name', 'mode' )
```

**Parameters**

<table>
<thead>
<tr>
<th>design-name</th>
<th>Name of the target design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Name of the mode that Database Designer should use when designing the database, one of the following:</td>
</tr>
</tbody>
</table>

- COMPREHENSIVE: Creates an initial or replacement design for all tables in the specified schemas. You typically create a comprehensive design for a new database.
• **INCREMENTAL**: Modifies an existing design with additional projection that are optimized for new or modified queries.

For more information, see [Design Types](#) in the Administrator's Guide.

**Privileges**

One of the following:

- Superuser
- Design creator

**Notes**

Incremental designs always inherit the K-safety value of the database.

**Examples**

The following examples show the two design mode options for the `VMART_DESIGN` design:

```sql
=> SELECT DESIGNER_SETDESIGN_TYPE('VMART_DESIGN', 'COMPREHENSIVE');
DESIGNER_SETDESIGN_TYPE
--------------------------
0
(1 row)

=> SELECT DESIGNER_SETDESIGN_TYPE('VMART_DESIGN', 'INCREMENTAL');
DESIGNER_SETDESIGN_TYPE
--------------------------
0
(1 row)
```

**See Also**

[Running Database Designer Programmatically](#)

**DESIGNER_SET_OPTIMIZATION_OBJECTIVE**

Valid only for comprehensive database designs, specifies the optimization objective Database Designer uses. Database Designer ignores this function for incremental designs.
DESIGNER_SET_OPTIMIZATION_OBJECTIVE stores the optimization objective in the DESIGNS table.

Behavior Type
Immutable

Syntax
DESIGNER_SET_OPTIMIZATION_OBJECTIVE ( 'design-name', 'policy' )

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>design-name</td>
<td>Name of the target design.</td>
</tr>
<tr>
<td>policy</td>
<td>Specifies the design's optimization policy, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• QUERY: Optimize for query performance. This can result in a larger</td>
</tr>
<tr>
<td></td>
<td>database storage footprint because additional projections might be</td>
</tr>
<tr>
<td></td>
<td>created.</td>
</tr>
<tr>
<td></td>
<td>• LOAD: Optimize for load performance so database size is minimized.</td>
</tr>
<tr>
<td></td>
<td>This can result in slower query performance.</td>
</tr>
<tr>
<td></td>
<td>• BALANCED: Balance the design between query performance and database size.</td>
</tr>
</tbody>
</table>

Privileges
One of the following:

• Superuser
• Design creator

Examples
The following example sets the optimization objective option for the VMART_DESIGN design:

```sql
=> SELECT DESIGNER_SET_OPTIMIZATION_OBJECTIVE( 'VMART_DESIGN', 'QUERY' );
DESIGNER_SET_OPTIMIZATION_OBJECTIVE
```
DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS

Specifies whether a design can include unsegmented projections. Vertica ignores this function on a one-node cluster, where all projections must be unsegmented.

Behavior Type
Immutable

Syntax
DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS ( 'design-name', unsegmented )

Parameters

<table>
<thead>
<tr>
<th>design-name</th>
<th>Name of the target design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsegmented</td>
<td>Boolean that specifies whether Database Designer can propose unsegmented projections for tables in this design. When you create a design, the propose_unsegmented_projections value in system table DESIGNS for this design is set to true. If DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS sets this value to false, Database Designer only proposes segmented projections for the design.</td>
</tr>
</tbody>
</table>

Privileges

One of the following:

- Superuser
- Design creator
Example

The following example specifies that Database Designer can propose only segmented projections for tables in the design VMART_DESIGN:

```sql
=> SELECT DESIGNER_SET_PROPOSE_UNSEGMENTED_PROJECTIONS('VMART_DESIGN', false);
```

See Also

Running Database Designer Programmatically

**DESIGNER_WAIT_FOR_DESIGN**

Waits for completion of operations that are populating and deploying the design. Ctrl+C cancels this operation and returns control to the user.

**Behavior Type**

Immutable

**Syntax**

```sql
DESIGNER_WAIT_FOR_DESIGN ( 'design-name' )
```

**Parameters**

| design-name | Name of the running design. |

**Privileges**

| Privileges |

One of the following:

- Superuser
- DBDUSER role with USAGE privilege on the design schema.
Examples

The following example requests to wait for the currently running design of VMART_DESIGN to complete:

```sql
=> SELECT DESIGNER_WAIT_FOR_DESIGN ('VMART_DESIGN');
```

See Also

- `DESIGNER_CANCEL_POPULATE_DESIGN`
- `DESIGNER_DROP_ALL_DESIGNS`
- `DESIGNER_DROP_DESIGN`
Database Management Functions

This section contains the database management functions specific to Vertica.

CLEAR_RESOURCE_REJECTIONS

Clears the content of the RESOURCE_REJECTIONS and DISK_RESOURCE_REJECTIONS system tables. Normally, these tables are only cleared during a node restart. This function lets you clear the tables whenever you need. For example, you might want to clear the system tables after you resolved a disk space issue that was causing disk resource rejections.

Syntax

CLEAR_RESOURCE_REJECTIONS();

Privileges

Superuser

Example

The following command clears the content of the RESOURCE_REJECTIONS and DISK_RESOURCE_REJECTIONS system tables:

```sql
=> SELECT clear_resource_rejections();
clear_resource_rejections
-------------------
   OK
(1 row)
```

See Also

- DISK_RESOURCE_REJECTIONS
- RESOURCE_REJECTIONS

CURRENT_SCHEMA

Returns the name of the current schema.
Behavior Type
Stable

Syntax
CURRENT_SCHEMA()

Note: You can call this function without parentheses.

Privileges
None

Examples
The following command returns the name of the current schema:

```sql
=> SELECT CURRENT_SCHEMA();
current_schema
--------------
public
(1 row)
```

The following command returns the same results without the parentheses:

```sql
=> SELECT CURRENT_SCHEMA;
current_schema
--------------
public
(1 row)
```

The following command shows the current schema, listed after the current user, in the search path:

```sql
=> SHOW SEARCH_PATH;
name | setting
--------------
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

See Also

- SET SEARCH_PATH
**DUMP_LOCKTABLE**

Returns information about deadlocked clients and the resources they are waiting for.

**Syntax**

```
DUMP_LOCKTABLE()
```

**Privileges**

None

**Notes**

Use DUMP_LOCKTABLE if Vertica becomes unresponsive:

1. Open an additional vsql connection.
2. Execute the query:

   ```
   => SELECT DUMP_LOCKTABLE();
   ```

   The output is written to vsql. See Monitoring the Log Files.

You can also see who is connected using the following command:

```
=> SELECT * FROM SESSIONS;
```

Close all sessions using the following command:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

Close a single session using the following command:

```
=> SELECT CLOSE_SESSION('session_id');
```

You get the session_id value from the V_MONITORSESSIONS system table.
See Also

- `CLOSE_ALL_SESSIONS`
- `CLOSE_SESSION`
- `LOCKS`
- `SESSIONS`

**DUMP_PARTITION_KEYS**

Dumps the partition keys of all projections in the system.

**Syntax**

```
DUMP_PARTITION_KEYS( )
```

**Note:** The ROS objects of partitioned tables without partition keys are ignored by the tuple mover and are not merged during automatic tuple mover operations.

**Privileges**

User must have select privileges on the table or usage privileges on the schema.

**Example**

```
=> SELECT DUMP_PARTITION_KEYS( );
Partition keys on node v_vmart_node0001
  Projection 'states_b0'
    Storage [ROS container]
      No of partition keys: 1
      Partition keys: NH
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: MA
  Projection 'states_b1'
    Storage [ROS container]
      No of partition keys: 1
      Partition keys: VT
    Storage [ROS container]
      No of partition keys: 1
      Partition keys: ME
    Storage [ROS container]
      No of partition keys: 1
```

Partition keys: CT

See Also

- DUMP_PROJECTION_PARTITION_KEYS
- DUMP_TABLE_PARTITION_KEYS
- PARTITION_PROJECTION
- PARTITION_TABLE
- PARTITIONS
- Partitioning Tables in the Administrator's Guide

HAS_ROLE

Indicates, with a Boolean value, whether a role has been assigned to a user. This function is useful for letting you check your own role membership.

Behavior Type

Stable

Syntax 1

HAS_ROLE( [ 'user_name' ] , 'role_name' );

Syntax 2

HAS_ROLE( 'role_name' );

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>[Optional] The name of a user to look up. Currently, only a superuser can supply the user_name argument.</td>
</tr>
<tr>
<td>role_name</td>
<td>The name of the role you want to verify has been granted.</td>
</tr>
</tbody>
</table>
Privileges

Users can check their own role membership by calling HAS_ROLE('role_name'), but only a superuser can look up other users' memberships using the optional user_name parameter.

Notes

You can query V_CATALOG system tables ROLES, GRANTS, and USERS to show any directly-assigned roles; however, these tables do not indicate whether a role is available to a user when roles may be available through other roles (indirectly).

Examples

User Bob wants to see if he has been granted the commentor role:

```sql
=> SELECT HAS_ROLE('commentor');
```

Output t for true indicates that Bob has been assigned the commentor role:

```
<table>
<thead>
<tr>
<th>HAS_ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

In the following function call, a superuser checks if the logadmin role has been granted to user Bob:

```sql
=> SELECT HAS_ROLE('Bob', 'logadmin');
```

```
<table>
<thead>
<tr>
<th>HAS_ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
```

To view the names of all roles users can access, along with any roles that have been assigned to those roles, query the V_CATALOG.ROLES system table. An asterisk in the output means role granted WITH ADMIN OPTION.

```sql
=> SELECT * FROM roles;
```

```
<table>
<thead>
<tr>
<th>role_id</th>
<th>name</th>
<th>assigned_roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273704964</td>
<td>public</td>
<td></td>
</tr>
<tr>
<td>45035996273704966</td>
<td>dbduser</td>
<td></td>
</tr>
<tr>
<td>45035996273704968</td>
<td>dbadmin</td>
<td>dbduser*</td>
</tr>
<tr>
<td>45035996273704972</td>
<td>pseudosuperuser</td>
<td>dbadmin*</td>
</tr>
<tr>
<td>45035996273704974</td>
<td>logreader</td>
<td></td>
</tr>
<tr>
<td>45035996273704976</td>
<td>logwriter</td>
<td></td>
</tr>
<tr>
<td>45035996273704978</td>
<td>logadmin</td>
<td>logreader,</td>
</tr>
</tbody>
</table>
```
See Also

- **GRANTS**
- **ROLES**
- **USERS**
- **Managing Users and Privileges**
- **Viewing a User's Role**

**KERBEROS_CONFIG_CHECK**

Tests the Kerberos configuration of a Vertica cluster. The function succeeds if it can access the keytab file and use it to kinit, and reports errors otherwise.

**Syntax**

```sql
KERBEROS_CONFIG_CHECK()
```

**Parameters**

This function has no parameters.

**Privileges**

This function does not require privileges.

**Examples**

The following example shows the results when the Kerberos configuration is valid.

```sql
=> SELECT KERBEROS_CONFIG_CHECK();
  kerberos_config_check
---------------------------------------------------------------------------------------------------------------------
  ok: krb5 exists at [/etc/krb5.conf]
  ok: Vertica Keytab file is set to [/etc/vertica.keytab]
  ok: Vertica Keytab file exists at [/etc/vertica.keytab]
```
Kerberos configuration parameters set in the database
KerberosServiceName: [vertica]
KerberosHostname: [data.hadoop.com]
KerberosRealm: [EXAMPLE.COM]
KerberosKeytabFile: [/etc/vertica.keytab]
Vertica Principal: [vertica/data.hadoop.com@EXAMPLE.COM]

ok: Vertica can kinit using keytab file

(1 row)

RUN_INDEX_TOOL

Runs the Index tool on a Vertica database to perform one of these tasks:

- Run a per-block cyclic redundancy check (CRC) on data storage to verify data integrity.
- Check that the sort order in ROS containers is correct.

The function writes summary information about its operation to standard output; detailed information on results is logged in vertica.log on the current node. For more about evaluating tool output, see:

- Evaluating CRC Errors
- Evaluating Sort Order Errors

You can also run the Index tool on a database that is down, from the Linux command line. For details, see CRC and Sort Order Check.

Caution: Use this function only under guidance from Vertica Support.

Syntax

RUN_INDEX_TOOL ( 'task', global, '[projection]' [, num-threads ] );

Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>Specifies the operation to run, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- checkcrc: Run a cyclic redundancy check (CRC) on each block of existing data storage to check the data integrity of ROS data blocks.</td>
</tr>
<tr>
<td></td>
<td>- checksort: Evaluate each ROS row to determine</td>
</tr>
</tbody>
</table>
whether it is sorted correctly. If ROS data is not sorted correctly in the projection's order, query results that rely on sorted data will be incorrect.

<table>
<thead>
<tr>
<th><strong>global</strong></th>
<th>Boolean, specifies whether to run the specified task on all nodes (true), or the current one (false).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projection</strong></td>
<td>Specifies the scope of the operation:</td>
</tr>
<tr>
<td></td>
<td>• Empty string (''): Run the check on all projections.</td>
</tr>
<tr>
<td></td>
<td>• A string that specifies one or more projections as follows:</td>
</tr>
<tr>
<td></td>
<td>• <em>proj-name</em>: Run the check on this projection</td>
</tr>
<tr>
<td></td>
<td>• <em>proj-prefix</em>&quot;: Run the check on all projections that begin with the string <em>proj-prefix</em>.</td>
</tr>
<tr>
<td><strong>num-threads</strong></td>
<td>An unsigned (positive) or signed (negative) integer that specifies the number of threads used to run this operation:</td>
</tr>
<tr>
<td></td>
<td>• (n): Number of threads, (\geq 1)</td>
</tr>
<tr>
<td></td>
<td>• (-n): Negative integer, denotes a fraction of all CPU cores as follows:</td>
</tr>
<tr>
<td></td>
<td>(\frac{\text{num-cores}}{n})</td>
</tr>
<tr>
<td></td>
<td>Thus, (-1) specifies all cores, (-2), half the cores, (-3), a third of all cores, and so on.</td>
</tr>
<tr>
<td></td>
<td>Default: (1)</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Optimizing Performance**

You can optimize meta-function performance by setting two parameters:

- **projection**: Narrows the scope of the operation to one or more projections.
- **num-threads**: Specifies the number of threads used to execute the function.
SET_CONFIG_PARAMETER

Specifies the value of a configuration parameter at the database level, or for a specific node.

**Important:** Vertica encourages use of ALTER NODE, ALTER DATABASE, and ALTER SESSION statements to set and clear configuration parameters. See Managing Configuration Parameters: VSQL for more information.

**Caution:** Vertica is designed to operate with minimal configuration changes, so use this capability sparingly. Carefully follow any documented guidelines for the parameter you wish to configure.

**Syntax**

`SET_CONFIG_PARAMETER( 'parameter-name', value, [ 'node-name' ] )`

**Parameters**

<table>
<thead>
<tr>
<th>parameter-name</th>
<th>The parameter value to set. See Configuration Parameters in the Administrator's Guide for a list of supported parameters, their purposes, and usage examples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value to set for <em>parameter-name</em>. Syntax for this argument varies depending upon the parameter and its expected data type. For strings, enclose the argument in single quotes; integer arguments can be unquoted. If value is specified as NULL, the parameter is cleared.</td>
</tr>
<tr>
<td>node-name</td>
<td>The name of the node whose parameter value you wish to set. If you omit this parameter or set it to NULL, the parameter is set at the database level. If a parameter is explicitly set for a node, the node setting supersedes the database-level setting.</td>
</tr>
</tbody>
</table>

**Note:** Some parameters require restart for the value to take effect.

**Privileges**

Superuser
Examples

The following examples show how to use SET_CONFIG_PARAMETER in various situations.

Set a Configuration Parameter at Database Level

Set the AnalyzeRowCountInterval parameter to 3600 at the database level:

```
=> SELECT SET_CONFIG_PARAMETER ('AnalyzeRowCountInterval',3600);
```

Find Details on All Configuration Parameters

Find all configuration parameters and information about them, including their current and default values:

```
=> SELECT * FROM CONFIGURATION_PARAMETERS;
```

See Also

- Managing Configuration Parameters: VSQL
- CONFIGURATION_PARAMETERS

SHUTDOWN

Forces a database to shut down, even if there are users connected.

Syntax

```
SHUTDOWN ( [ 'false' | 'true' ])
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>false</strong></td>
<td>[Default] Returns a message if users are connected. Has the same effect as supplying no parameters.</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>Performs a moveout operation and forces the database to shut down, disallowing further connections.</td>
</tr>
</tbody>
</table>

Privileges

Superuser
Notes

- Quotes around the true or false arguments are optional.
- Issuing the shutdown command without arguments or with the default (false) argument returns a message if users are connected, and the shutdown fails. If no users are connected, the database performs a moveout operation and shuts down.
- Issuing the SHUTDOWN('true') command forces the database to shut down whether users are connected or not.
- You can check the status of the shutdown operation in the vertica.log file:

  2010-03-09 16:51:52.625 unknown:0x7fc6d6d2e700
  [Init] <INFO> Shutdown complete. Exiting.

- As an alternative to SHUTDOWN(), you can also temporarily set MaxClientSessions to 0 and then use CLOSE_ALL_SESSIONS(). New client connections cannot connect unless they connect using the dbadmin account. See CLOSE_ALL_SESSIONS for details.

Examples

The following command attempts to shut down the database. Because users are connected, the command fails:

```
=> SELECT SHUTDOWN('false');
NOTICE: Cannot shut down while users are connected
```

SHUTDOWN() and SHUTDOWN('false') perform the same operation:

```
=> SELECT SHUTDOWN();
NOTICE: Cannot shut down while users are connected
```

Using the 'true' parameter forces the database to shut down, even though clients might be connected:

```
=> SELECT SHUTDOWN('true');
```
Shutdown: moveout complete
(1 row)

See Also

- SESSIONS

Directed Queries Functions

The following meta-functions let you batch export query plans as directed queries from one Vertica database, and import those directed queries to another database.

**EXPORT_DIRECTED_QUERIES**

Generates SQL for creating directed queries from a set of input queries, and writes the SQL to the specified file or to standard output.

**Syntax**

```
EXPORT_DIRECTED_QUERIES('input-file', '[output-file]')
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-file</td>
<td>A SQL file that contains one or more input queries. See <strong>Input Format</strong> below for details on format requirements.</td>
</tr>
<tr>
<td>output-file</td>
<td>Specifies where to write the generated SQL for creating directed queries. If the file name already exists, <strong>EXPORT_DIRECTED_QUERIES</strong> returns with an error. If you supply an empty string, Vertica writes the SQL to standard output. See <strong>Output Format</strong> below for details.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser
Input Format

The input file that you supply to EXPORT_DIRECTED_QUERIES contains one or more input queries. For each input query, you can optionally specify two fields that are used in the generated directed query:

- **DirQueryName** provides the directed query's unique identifier, a string that conforms to conventions described in Identifiers.

- **DirQueryComment** specifies a quote-delimited string, up to 128 characters.

You format each input query as follows:

```plaintext
--DirQueryName=query-name
--DirQueryComment='comment'
input-query
```

Output Format

EXPORT_DIRECTED_QUERIES generates SQL for creating directed queries, and writes the SQL to the specified file or to standard output. In both cases, output conforms to the following format:

```sql
/* Query: directed-query-name */
/* Comment: directed-query-comment */
SAVE QUERY input-query;
CREATE DIRECTED QUERY CUSTOM 'directed-query-name'
COMMENT 'directed-query-comment'
OPTVER 'vertica-release-num'
PSDATE 'timestamp'
annotated-query
```

Error Handling

If any errors or warnings occur during EXPORT_DIRECTED_QUERIES execution, it returns with a message like this one:

```plaintext
1 queries successfully exported.
1 warning message was generated.
Queries exported to /home/dbadmin/outputQueries.
See error report, /home/dbadmin/outputQueries.err for details.
```

EXPORT_DIRECTED_QUERIES writes all errors and warnings to a file that it creates on the same path as the output file, and uses the output file's base name.

For example:
WARNING: Name field not supplied. Using auto-generated name: 'Autoname:2016-04-25 15:03:32.115317.0'
Input Query: SELECT employee_dimension.employee_first_name, employee_dimension.employee_last_name, employee_dimension.job_title FROM public.employee_dimension WHERE (employee_dimension.employee_city = 'Boston':varchar(6)) ORDER BY employee_dimension.job_title;
END WARNING

Examples

See Batch Query Plan Export in the Administrator's Guide.

See Also

- Batch Query Plan Export
- IMPORT_DIRECTED_QUERIES

IMPORT_DIRECTED_QUERIES

Imports to the database catalog directed queries from a SQL file that was generated by EXPORT_DIRECTED_QUERIES. If no directed queries are specified, Vertica lists all directed queries in the SQL file.

Syntax

IMPORT_DIRECTED_QUERIES( 'export-file'[, 'directed-query-name[,...]' ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>export-file</td>
<td>A SQL file generated by EXPORT_DIRECTED_QUERIES. When you run this file, Vertica creates the specified directed queries in the current database catalog.</td>
</tr>
<tr>
<td>directed-query-name</td>
<td>The name of a directed query that is defined in export-file. You can specify multiple comma-delimited directed query names. If you omit this parameter, Vertica lists the names of all directed queries in export-file.</td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x)
Privileges
Superuser

See Also
- Batch Query Plan Export
- EXPORT_DIRECTED_QUERIES

Eon Functions
The following functions are meant to be used in Eon Mode.

ALTER_LOCATION_SIZE
Changes the size of the depot location on one node or all of the nodes in the database.
Decreasing the depot size can result in evictions if they are necessary to make space available.
Evictions occur when Vertica removes data from the depot.

Syntax
ALTER_LOCATION_SIZE( path, node_name, new_size)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Where Vertica stores the depot. The type of filesystem on which the location is based determines the format of this argument. For depot locations on the Linux filesystem, path must be an absolute path to the directory where Vertica writes the depot's data.</td>
</tr>
<tr>
<td>node_name</td>
<td>The node on which you want to alter the depot's size. To alter the depot size of all nodes, you must pass an empty string.</td>
</tr>
<tr>
<td>new_size</td>
<td>The new size the depot:</td>
</tr>
<tr>
<td></td>
<td>- integer%: Percentage of total disk space</td>
</tr>
</tbody>
</table>
• **integer**{K|M|G|T}: Amount of disk space in kilobytes, megabytes, gigabytes, or terabytes.

  Default Value: 80% of the disk space on the node. If it was not set at creation time.

**Privileges**

Superuser

**Examples**

Change the depot size on all the nodes in the database to 60 gigabytes:

```
=> SELECT ALTER_LOCATION_SIZE('/home/dbadmin/depot/', '', '60G');

ALTER_LOCATION_SIZE

/home/dbadmin/depotSize changed.
(1 row)
```

Change the depot size of one node in the database to 50% of the total disk space on the node:

```
=> SELECT ALTER_LOCATION_SIZE('/home/dbadmin/depot/', 'v_eon_walkthrough_db_node0001', '50%');

ALTER_LOCATION_SIZE

/home/dbadmin/depotSize changed.
(1 row)
```

**See Also**

• [Depot](#)

**CLEAR_DATA_DEPOT**

Deletes the specified data from depots in the database. If you use a table name with the function, then the data for that table is deleted from the depots in the database.

**Syntax**

`CLEAR_DATA_DEPOT( [ table_name ] )`
Parameters

| table_name   | (Optional) The name of the table to delete from all of the depots in the cluster. |

Privileges

Superuser

Examples

Delete all of the data in the depots in the database:

```
=> SELECT CLEAR_DATA_DEPOT();
CLEAR_DATA_DEPOT
-------------------
Cache cleared
(1 row)
```

Delete the User_Experience_Data table from all of the depots in the cluster:

```
=> SELECT CLEAR_DATA_DEPOT('User_Experience_Data');
CLEAR_DATA_DEPOT
-------------------
Cache cleared
(1 row)
```

See Also

- Depot
- Eon Database Overview

**FINISH_FETCHING_FILES**

Fetches all the files that are queued for fetching from the communal storage.

Syntax

```
FINISH_FETCHING_FILES()
```
Privileges
Superuser

Examples

Get all the files queued from communal storage:

```sql
=> SELECT FINISH_FETCHING_FILES();
FINISH_FETCHING_FILES
-------------------------
Finished fetching all the files
(1 row)
```

See Also

- Eon Database Overview

**FLUSH_REAPER_QUEUE**

Deletes all of the data marked for deletion in the database. Use this function when you want to remove all of the data marked for deletion before the reaper performs its task.

Syntax

```sql
FLUSH_REAPER_QUEUE()
```

Privileges
Superuser

Examples

Removes all of the files that have been marked for deletion in the database:

```sql
=> SELECT FLUSH_REAPER_QUEUE();
FLUSH_REAPER_QUEUE
-------------------------------
Finished deleting all the files in the reaper queue
(1 row)
```
REBALANCE_SHARDS

Rebalances shard assignments across the cluster. If the current session ends, the operation immediately aborts.

Syntax

REBALANCE_SHARDS()

Privileges

Superuser

Examples

The following example rebalances the shards of all the nodes in the database:

```sql
=> SELECT REBALANCE_SHARDS();
REBALANCE_SHARDS
------------------------
REBALANCED_SHARDS
(1 row)
```

After running REBALANCE_SHARDS check the subscription status by querying the SESSION_SUBSCRIPTIONS table, for example:

```sql
=> SELECT * from session_subscriptions where is_participating='t' order by node_name, shard_name;
```

See Also

- [Shards and Subscriptions](#)
- [Eon Database Overview](#)
SYNC_CATALOG

Immediately synchronizes the catalog to shared storage to allow revive as of the current catalog version in the case of an imminent crash. When you run sync_catalog, Vertica synchronizes all pending Checkpoint and Transaction Logs to shared storage.

Syntax

SYNC_CATALOG( [ 'node_name' ] )

Parameters

| node_name | (Optional) The name of the node you want to synchronize. |

Privileges

Superuser

Examples

Synchronize the catalog on all nodes:

=> SELECT SYNC_CATALOG();

Synchronize the catalog on a specific node:

=> SELECT SYNC_CATALOG( 'node001' );
Epoch Management Functions

This section contains the epoch management functions specific to Vertica.

ADVANCE_EPOCH

Manually closes the current epoch and begins a new epoch.

Syntax

ADVANCE_EPOCH ( [ integer ] )

Parameters

| integer | Specifies the number of epochs to advance. |

Privileges

Superuser

Notes

This function is primarily maintained for backward compatibility with earlier versions of Vertica.

Example

The following command increments the epoch number by 1:

```sql
=> SELECT ADVANCE_EPOCH(1);
```

GET_AHM_EPOCH

Returns the number of the epoch in which the Ancient History Mark is located. Data deleted up to and including the AHM epoch can be purged from physical storage.
Syntax

GET_AHM_EPOCH()

Note: The AHM epoch is 0 (zero) by default (purge is disabled).

Privileges
None

Examples

=> SELECT GET_AHM_EPOCH();

Current AHM epoch: 0
(1 row)

GET_AHM_TIME

Returns a TIMESTAMP value representing the Ancient History Mark. Data deleted up to and including the AHM epoch can be purged from physical storage.

Syntax

GET_AHM_TIME()

Privileges
None

Examples

=> SELECT GET_AHM_TIME();

(1 row)
GET_CURRENT_EPOCH

The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.

Returns the number of the current epoch.

Syntax
GET_CURRENT_EPOCH()

Privileges
None

Examples

```sql
=> SELECT GET_CURRENT_EPOCH();
   GET_CURRENT_EPOCH
-----------------
       683
(1 row)
```

GET_LAST_GOOD_EPOCH

Returns the last good epoch number. If the database has no projections, the function returns an error.

Syntax
GET_LAST_GOOD_EPOCH()

Privileges
None
Examples

```sql
=> SELECT GET_LAST_GOOD_EPOCH();
GET_LAST_GOOD_EPOCH
---------------------
  682
(1 row)
```

**MAKE_AHM_NOW**

Sets the Ancient History Mark (AHM) to the greatest allowable value. This lets you purge all deleted data.

Caution: After running this function, you cannot query historical data that precedes the current epoch. Only database administrators should use this function.

MAKE_AHM_NOW performs the following operations:

- Advances the epoch.
- Performs a moveout operation on all projections.
- Sets the AHM to the last good epoch (LGE) — at least to the epoch that is current when you execute MAKE_AHM_NOW.

**Syntax**

```sql
MAKE_AHM_NOW ( [ true ] )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Allows AHM to advance when one of the conditions is true:</td>
</tr>
<tr>
<td></td>
<td>- One or more nodes are down.</td>
</tr>
<tr>
<td></td>
<td>- One projection is being refreshed from another (retentive refresh).</td>
</tr>
</tbody>
</table>

In either case, you must supply this argument to MAKE_AHM_NOW, otherwise Vertica returns an error. If you execute MAKE_AHM_NOW(true) during retentive refresh, Vertica rolls back the refresh operation and advances the AHM.
Privileges
Superuser

Setting AHM When Nodes Are Down

If you run MAKE_AHM_NOW while any node is down, you must supply an argument of true; otherwise, Vertica returns an error. In the following example, MAKE_AHM_NOW advances the AHM even though a node is down:

```sql
=> SELECT MAKE_AHM_NOW(true);
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in get cluster LGE
WARNING: Received no response from v_vmartdb_node0002 in set AHM
MAKE_AHM_NOW
AHM set (New AHM Epoch: 684)
(1 row)
```

Caution: If the AHM is advanced beyond the last good epoch of the failed nodes, those nodes must recover all data from scratch.

See Also
- `SET_AHM_EPOCH`
- `SET_AHM_TIME`

**SET_AHM_EPOCH**

Sets the Ancient History Mark (AHM) to the specified epoch. This function allows deleted data up to and including the AHM epoch to be purged from physical storage.

`SET_AHM_EPOCH` is normally used for testing purposes. Instead, consider using `SET_AHM_TIME` which is easier to use.

**Syntax**

```
SET_AHM_EPOCH ( epoch, [ true ])
```
Parameters

**epoch**  
Specifies one of the following:
- The number of the epoch in which to set the AHM
- Zero (0) (the default) disables PURGE

**true**  
Allows the AHM to advance when nodes are down.

Note: If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. Use with care.

Privileges

Superuser

Restrictions

The number of the specified epoch must be:
- Greater than the current AHM epoch
- Less than the current epoch
- Less than or equal to the cluster last good epoch (the minimum of the last good epochs of the individual nodes in the cluster)

Use the **SYSTEM** table to see current values of various epochs related to the AHM, For example:

```sql
=> SELECT * from SYSTEM;
-[ RECORD 1 ]-------------------------------------------
current_timestamp | 2009-08-11 17:09:54.651413
current_epoch     | 1512
ahm_epoch         | 961
last_good_epoch   | 1510
refresh_epoch     | -1
designed_fault_tolerance | 1
node_count        | 4
node_down_count   | 0
current_fault_tolerance | 1
catalog_revision_number | 1590
wos_used_bytes    | 0
wos_row_count     | 0
```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ros_used_bytes</td>
<td>41490783</td>
</tr>
<tr>
<td>ros_row_count</td>
<td>1298104</td>
</tr>
<tr>
<td>total_used_bytes</td>
<td>41490783</td>
</tr>
<tr>
<td>total_row_count</td>
<td>1298104</td>
</tr>
</tbody>
</table>

All nodes must be up. You cannot use SET_AHM_EPOCH when any node in the cluster is down, except by using the optional true parameter.

When a node is down and you use MAKE_AHM_NOW, the following error is printed to the vertica.log:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

Examples

The following command sets the AHM to a specified epoch of 12:

```
=> SELECT SET_AHM_EPOCH(12);
```

The following command sets the AHM to a specified epoch of 2 and allows the AHM to advance despite a failed node:

```
=> SELECT SET_AHM_EPOCH(2, true);
```

See Also

- MAKE_AHM_NOW
- SET_AHM_TIME
- SYSTEM

SET_AHM_TIME

Sets the Ancient History Mark (AHM) to the epoch corresponding to the specified time on the initiator node. This function allows historical data up to and including the AHM epoch to be purged from physical storage.

Syntax

```
SET_AHM_TIME ( time , [ true ] )
```
Parameters

| time | Is a `TIMESTAMP/TIMESTAMPZ` value that is automatically converted to the appropriate epoch number. |
| true | [Optional] Allows the AHM to advance when nodes are down. Note: If the AHM is advanced after the last good epoch of the failed nodes, those nodes must recover all data from scratch. |

Privileges

Superuser

Notes

- `SET_AHM_TIME` returns a `TIMESTAMP WITH TIME ZONE` value representing the end point of the AHM epoch.
- You cannot change the AHM when any node in the cluster is down, except by using the optional `true` parameter.
- When a node is down and you issue `SELECT MAKE_AHM_NOW()`, the following error is printed to the vertica.log:

```
Some nodes were excluded from setAHM. If their LGE is before the AHM they will perform full recovery.
```

Examples

Epochs depend on a configured epoch advancement interval. If an epoch includes a three-minute range of time, the purge operation is accurate only to within minus three minutes of the specified timestamp:

```sql
=> SELECT SET_AHM_TIME('2008-02-27 18:13');
set_ahm_time
---------------------
AHM set to '2008-02-27 18:11:50-05'
(1 row)
```
Note: The –05 part of the output string is a time zone value, an offset in hours from UTC (Universal Coordinated Time, traditionally known as Greenwich Mean Time, or GMT).

In the previous example, the actual AHM epoch ends at 18:11:50, roughly one minute before the specified timestamp. This is because SET_AHM_TIME selects the epoch that ends at or before the specified timestamp. It does not select the epoch that ends after the specified timestamp because that would purge data deleted as much as three minutes after the AHM.

For example, using only hours and minutes, suppose that epoch 9000 runs from 08:50 to 11:50 and epoch 9001 runs from 11:50 to 15:50. SET_AHM_TIME( '11:51' ) chooses epoch 9000 because it ends roughly one minute before the specified timestamp.

In the next example, suppose that a node went down at 11:00:00 AM on January 1st 2017. At noon, you want to advance the AHM to 11:15:00, but the node is still down.

Suppose you try to set the AHM using this command:

```sql
=> SELECT SET_AHM_TIME('2017-01-01 11:15:00');
```

Then you will receive an error message. Vertica prevents you from moving the AHM past the point where a node went down. Vertica returns this error to prevent the AHM from advancing past the down node's last good epoch. You can force the AHM to advance by supplying the optional second parameter:

```sql
=> SELECT SET_AHM_TIME('2017-01-01 11:15:00', true);
```

However, if you force the AHM past the last good epoch, the failed node will have to recover from scratch.

See Also

- MAKE_AHM_NOW
- SET_AHM_EPOCH
- SET_DATESTYLE
- TIMESTAMP/TIMESTAMPTZ

**Flex Table Functions**

This section contains helper functions for use in working with flex tables.
**Note:** While the functions are available to all users, they are applicable only to flex table, their associated `flex_table_keys` table and `flex_table_view` views. By computing keys and creating views from flex table data, the functions facilitate SELECT queries. One function restores the original keys table and view that were made when you first created the flex table. For more information, see Using Flex Tables.

**BUILD_FLEXTABLE_VIEW**

Creates, or re-creates, a view for a default or user-defined `_keys` table, ignoring any empty keys.

**Syntax**

```
BUILD_FLEXTABLE_VIEW('[[database.]schema.]flex-table' [,,'view-name'] [,,'user-keys-table'] )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>flex-table</code></td>
<td>The flex table name. By default, this function builds or rebuilds a view for the input table with the current contents of the associated <code>flex_table_keys</code> table.</td>
</tr>
<tr>
<td><code>view-name</code></td>
<td>A custom view name. Use this option to build a new view for <code>flex-table</code> with the name you specify.</td>
</tr>
<tr>
<td><code>user-keys-table</code></td>
<td>Specifies a keys table from which to create the view. Use this option if you created a custom <code>user-keys</code> table from the flex table map data, rather than from the default <code>flex_table_keys</code> table. The function builds a view from the keys in <code>user-keys</code> table, rather than from the <code>flex_table_keys</code> table.</td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how to call `build_flextable_view` with 1, 2, or 3 arguments.
Creating a Default View

To create, or re-create, a default view:

1. Call the function with an input flex table, `darkdata`:

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata');

The view public.darkdata_view is ready for querying
(1 row)
```

The function creates a view with the default name (darkdata_view) from the `darkdata_keys` table.

2. Query a key name (`user.id`) from the new or updated view:

```sql
=> SELECT "user.id" FROM darkdata_view;

user.id
---------
340857907
727774963
390498773
288187825
164464905
125434448
601328899
352494946
(12 rows)
```

Creating a Custom Name View

To create, or re-create, a view with a custom name:

1. Call the function with two arguments, an input flex table, `darkdata`, and the name of the view to create, `dd_view`:

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view');

The view public.dd_view is ready for querying
(1 row)
```

2. Query a key name (`user.lang`) from the new or updated view (dd_view):

```sql
=> SELECT "user.lang" FROM dd_view;

user.lang
---------
tr
en
es
```
Creating a View from a Custom Keys Table

To create a view from a custom _keys table with build_flextable_view, the custom table must have the same schema and table definition as the default table (darkdata_keys).

Create a custom keys table, using any of these three approaches:

1. Create a columnar table with all keys from the default keys table for a flex table (darkdata_keys):

   ```sql
   => CREATE TABLE new_darkdata_keys AS SELECT * FROM darkdata_keys;
   ```

2. Create a columnar table without content (LIMIT 0) from the default keys table for a flex table (darkdata_keys):

   ```sql
   CREATE TABLE new_darkdata_keys AS SELECT * FROM darkdata_keys LIMIT 0;
   ```

   ```sql
   kdb=> SELECT * FROM new_darkdata_keys;
   key_name | frequency | data_type_guess
   ===========
   ===========
   (0 rows)
   ```

3. Create a columnar table without content (LIMIT 0) from the default keys table, and insert two values ('user.lang', 'user.name') into the key_name column:

   ```sql
   => CREATE TABLE dd_keys AS SELECT * FROM darkdata_keys LIMIT 0;
   CREATE TABLE
   => INSERT INTO dd_keys (key_name) values ('user.lang');
   OUTPUT
   -------
   1
   (1 row)
   => INSERT INTO dd_keys (key_name) values ('user.name');
   OUTPUT
   -------
   1
   (1 row)
   => SELECT * FROM dd_keys;
   key_name | frequency | data_type_guess
   ===========
   ===========
   user.lang | |
4. After creating a custom keys table, call `build_flextable_view` with all arguments (an input flex table, the new view name, the custom keys table):

```sql
=> SELECT BUILD_FLEXTABLE_VIEW('darkdata', 'dd_view', 'dd_keys');
build_flextable_view
---------------------------------------------
The view public.dd_view is ready for querying
(1 row)
```

5. Query the new view:

```sql
=> SELECT * FROM dd_view;
```

See Also

- `COMPUTE_FLEXTABLE_KEYS`
- `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`
- `MATERIALIZE_FLEXTABLE_COLUMNS`
- `RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW`

**COMPUTE_FLEXTABLE_KEYS**

Computes the virtual columns (keys and values) from the flex table VMap data. Use this function to compute keys without creating an associated table view. To also build a view, use `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`.

The function stores its results in the associated flex _keys table, which has the following columns:

- `key_name`
- `frequency`
- `data_type_guess`

For more information, see Computing Flex Table Keys.
Syntax

```
COMPUTE_FLEXTABLE.Keys('[[database.]schema.]flex-table')
```

Arguments

| [database.]schema | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:  
| myschema.thisDbObject | If you specify a database, it must be the current database. |

| flex-table | The name of a flex table. |

Using Data Type Guessing

The results of the flex_keys table data_type_guess column depend on the EnableBetterFlexTypeGuessing configuration parameter. By default, the parameter is 1 (ON). This setting results in the function returning all non-string keys in the data_type_guess column as one of the following types (and others listed in SQL Data Types):

- BOOLEAN
- INTEGER
- FLOAT
- TIMESTAMP
- DATE

Setting the configuration parameter to 0 (OFF), results in the function returning only string types ([LONG]VARCHAR) or ([LONG] VARBINARY) for all values in the data_type_guess column of the flex_keys table.

Assigning Flex Key Data Types

Use the sample CSV data in this section to compare the results of using or not using the EnableBetterFlexTypeGuessing configuration parameter. When the parameter is ON,
the function determines key non-string data types in your map data more accurately. The default for the parameter is 1 (ON).

To compare the data type assignment results, complete the following steps:

1. Save the CSV data file (here, as trees.csv).

2. Create a flex table (trees) and load trees.csv using the fcsvparser:

```
=> CREATE FLEX TABLE trees();
=> COPY trees FROM '/home/dbadmin/tempdat/trees.csv' PARSER fcsvparser();
```

3. Use COMPUTE_FLEXTABLE_KEYS with the trees flex table.

```
=> SELECT COMPUTE_FLEXTABLE_KEYS('trees');
COMPUTE_FLEXTABLE_KEYS
---------------------------------------------
Please see public.trees_keys for updated keys
(1 row)
```

4. Query the trees_keys table output:

```
=> SELECT * FROM trees_keys;
  key_name | frequency | data_type_guess
----------|-----------|------------------
 Year      | 6         | Integer          
 Quarter   | 6         | Integer          
 Region    | 6         | Varchar(66)      
 Available | 6         | Boolean          
 Number of Quotes | 6 | Integer          
 Grade     | 6         | Varchar(20)      
 Species   | 6         | Varchar(22)      
 Pond Value | 6      | Numeric(8,3)     
(8 rows)
```

5. Set the EnableBetterFlexTypeGuessing parameter to 0 (OFF).

6. Call COMPUTE_FLEXTABLE_KEYS with the trees flex table again.

7. Query the trees_keys table to compare the data_type_guess values with the previous results. Without the configuration parameter set, all of the non-string data types are VARCHARS of various lengths:
```sql
SELECT *
FROM trees_keys;
```

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Quarter</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Region</td>
<td>6</td>
<td>varchar(66)</td>
</tr>
<tr>
<td>Available</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Grade</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Number of Quotes</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Pond Value</td>
<td>6</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>Species</td>
<td>6</td>
<td>varchar(22)</td>
</tr>
</tbody>
</table>

(8 rows)

8. To maintain accurate results for non-string data types, set the EnableBetterFlexTypeGuessing parameter back to 1 (ON).

For more information about setting the EnableBetterFlexTypeGuessing configuration parameter, see Setting Flex Table Configuration Parameters.

See Also

- **BUILD_FLEXTABLE_VIEW**
- **COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**
- **MATERIALIZE_FLEXTABLE_COLUMNS**
- **RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW**

**COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW**

Combines the functionality of **BUILD_FLEXTABLE_VIEW** and **COMPUTE_FLEXTABLE_KEYS** to compute virtual columns (keys) from the VMap data of a flex table and construct a view. Creating a view with this function ignores empty keys. If you do not need to perform both operations together, use one of the single-operation functions instead.

**Syntax**

```sql
COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW( 'flex_table' )
```

**Arguments**

| flex_table | The name of a flex table. |
Examples

This example shows how to call the function for the darkdata flex table.

```sql
=> SELECT COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW('darkdata');
              compute_flextable_keys_and_build_view
-------------------------------------------------------------------
  Please see public.darkdata_keys for updated keys
The view public.darkdata_view is ready for querying
(1 row)
```

See Also

- BUILD_FLEXTABLE_VIEW
- COMPUTE_FLEXTABLE_KEYS
- MATERIALIZE_FLEXTABLE_COLUMNS
- RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

EMPTYMAP

Constructs a new VMap with one row but without keys or data. Use this transform function to populate a map without using a flex parser. Instead, you use either from SQL queries or from map data present elsewhere in the database.

Syntax

EMPTYMAP()

Arguments

None

Examples

Create an Empty Map

```sql
=> SELECT EMPTYMAP();
  emptymap
-------------------------------------------------------------------
```
Create an Empty Map from an Existing Flex Table

If you create an empty map from an existing flex table, the new map has the same number of rows as the table from which it was created.

This example shows the result if you create an empty map from the `darkdata` table, which has 12 rows of JSON data:

```
=> SELECT EMPTYMAP() FROM darkdata;
emptymap
-----------------------------
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
\01\00\00\00\00\04\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
(12 rows)
```

See Also

- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
- `MAPSIZE`
- `MAPTOSTRING`
MAPVALUES

MAPVERSION

MAPAGGREGATE

Returns a LONG VARBINARY VMap with keys and value pairs supplied from two VARCHAR input columns of an existing columnar table. Using this function requires specifying an over() clause for the source table.

Syntax

MAPAGGREGATE(source_column1, source_column2)

Arguments

<table>
<thead>
<tr>
<th>source_column1</th>
<th>Table column with values to use as the keys of the key/value pair of the returned VMap data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_column2</td>
<td>Table column with values to use as the values in the key/value pair of the returned VMap data.</td>
</tr>
</tbody>
</table>

Examples

This example creates a columnar table btest, with two VARCHAR columns, named keys and values, and adds three sets of values:

```sql
=> CREATE TABLE btest(keys varchar(10), values varchar(10));
CREATE TABLE
=> COPY btest FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
   >> one|1
   >> two|2
   >> three|3
   >> \.  
```

After populating the btest table, call mapaggregate(), using the the over (PARTITION BEST) clause. This call returns the raw_map data:

```sql
=> SELECT MAPAGGREGATE(keys, values) OVER(PARTITION BEST) FROM btest;
raw_map
------------------------------------------------------------------------------------------------------------------
```
The next example illustrates using `MAPTOSTRING()` with the returned `raw_map` from `mapaggregate()` to see the values:

```sql
-> SELECT MAPTOSTRING(raw_map) FROM (SELECT MAPAGGREGATE(keys, values) OVER(PARTITION BEST) FROM btest) bit;

 maptostream
------------------------
 {  "one": "1",
    "three": "3",
    "two": "2"
 }
(1 row)
```

See Also

- `EMPTYMAP`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
- `MAPSIZE`
- `MAPTOSTRING`
- `MAPVALUES`
- `MAPVERSION`

**MAPCONTAINSKEY**

Determines whether a VMap contains a virtual column (key). This scalar function returns true (t), if the virtual column exists, or false (f) if it does not. Determining that a key exists before
calling maplookup() lets you distinguish between NULL returns. The maplookup() function uses for both a non-existent key and an existing key with a NULL value.

Syntax

MAPCONTAINSKEY(VMap_data, 'virtual_column_name')

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
</tbody>
</table>

| virtual_column_name | The name of the key to check. |

Examples

This example shows how to use the mapcontainskey() functions with maplookup(). View the results returned from both functions. Check whether the empty fields that maplookup() returns indicate a NULL value for the row (t) or no value (f):

You can use mapcontainskey() to determine that a key exists before calling maplookup(). The maplookup() function uses both NULL returns and existing keys with NULL values to indicate a non-existent key.

```
=> SELECT MAPLOOKUP(__raw__, 'user.location'), MAPCONTAINSKEY(__raw__, 'user.location')
FROM darkdata ORDER BY 1;
maplookup | mapcontainskey
-----------+-------------------
   t       |       t           |
   t       |       t           |
   t       |       t           |
   t       |       t           |
    t      |       t           |
    t      |       t           |
    t      |       t           |
    t      |       t           |
    t      |       t           |
```
See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPTOSTRING
- MAPVALUES
- MAPVERSION

MAPCONTAINSVALUE

Determines whether a VMap contains a specific value. Use this scalar function to return true (t), if the value exists, or false (f), if it does not.

Syntax

MAPCONTAINSVALUE(VMap_data, 'virtual_column_value')

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>- Data returned from a map function such as maplookup()</td>
</tr>
</tbody>
</table>
Vertica Documentation
SQL Reference Manual

l

virtual_column_value

Other database content

The value whose existence you want to confirm.

Examples
This example shows how to use mapcontainsvalue() to determine whether or not a virtual
column contains a particular value. Create a flex table (ftest), and populate it with some
virtual columns and values. Name both virtual columns one:
=> CREATE FLEX TABLE ftest();
CREATE TABLE
=> copy ftest from stdin parser fjsonparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> {"one":1, "two":2}
>> {"one":"one","2":"2"}
>> \.

Call mapcontainsvalue() on the ftest map data. The query returns false (f) for the first
virtual column, and true (t) for the second , which contains the value one:
=> SELECT MAPCONTAINSVALUE(__raw__, 'one') FROM ftest;
mapcontainsvalue
-----------------f
t
(2 rows)

See Also
l

EMPTYMAP

l

MAPAGGREGATE

l

MAPCONTAINSKEY

l

MAPITEMS

l

MAPKEYS

l

MAPKEYSINFO

l

MAPLOOKUP

l

MAPSIZE

Vertica Analytic Database (9.0.x)

Page 3074 of 6180


MAPDELIMITEDEXTRACTOR

Extracts data with a delimiter character, and other optional arguments, returning a single VMap value. The USING PARAMETERS phrase specifies optional parameters for the function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delimiter</td>
<td>VARCHAR</td>
<td>Single delimiter character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>header_names</td>
<td>VARCHAR</td>
<td>[Optional] Specifies header names for columns.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> ucoln</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where n is the column offset number, starting with 0 for the first column. The function uses default values if you do not specify values for the header_names parameter.</td>
</tr>
<tr>
<td>trim</td>
<td>BOOLEAN</td>
<td>[Optional] Trims white space from header names and field values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
<tr>
<td>treat_empty_val_as_null</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies that empty fields become NULLs, rather than empty strings (' ').</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default value:</strong> true</td>
</tr>
</tbody>
</table>

Examples

These examples use a short set of delimited data:

<table>
<thead>
<tr>
<th>Name</th>
<th>CITY</th>
<th>New city</th>
<th>State</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>BOSTON</td>
<td>boston</td>
<td>MA</td>
<td>01</td>
</tr>
<tr>
<td>Eric</td>
<td>Burlington</td>
<td>BURLINGTON</td>
<td>MA</td>
<td>02</td>
</tr>
<tr>
<td>Jamie</td>
<td>cambridge</td>
<td>CAMBRIDGE</td>
<td>MA</td>
<td>08</td>
</tr>
</tbody>
</table>
To begin, save this data as `delim.dat`.

1. Create a flex table, `dflex`:

```
=> CREATE FLEX TABLE dflex();
```  

2. Use COPY to load the `delim.dat` file. Use the flex tables `fdelimitedparser` with the `header='false'` option:

```
=> COPY dflex FROM '/home/release/kmm/flextables/delim.dat' parser fdelimitedparser (header='false');
  Rows Loaded
  4
  (1 row)
```  

3. Create a columnar table, `dtab`, with an identity `id` column, a `delim` column, and a column to hold a VMap, named `vmap`:

```
=> CREATE TABLE dtab (id IDENTITY(1,1), delim varchar(128), vmap long varbinary(512));
```  

4. Use COPY to load the `delim.dat` file into the `dtab` table. For the `mapdelimitedextractor` function, add a header row with `USING PARAMETERS header_names=` option to specify the header row for the sample data, along with `delimiter '!'`:

```
=> COPY dtab(delim, vmap AS MAPDELIMITEDEXTRACTOR (delim
   USING PARAMETERS header_names='Name|CITY|New City|State|Zip')) FROM
  '/home/dbadmin/data/delim.dat'
  DELIMITER '!';
  Rows Loaded
  4
  (1 row)
```  

5. Use `maptostring` for the flex table `dflex` to view the `__raw__` column contents. Notice the default header names in use (`ucol0 – ucol4`), since you specified `header='false'` when you loaded the flex table:

```
=> SELECT MAPTOSTRING(__raw__) FROM dflex limit 10;
```

```
{  
  "ucol0" : "Jamie",
  "ucol1" : "cambridge",
  "ucol2" : "CAMBRIDGE",
```
6. Use maptostring again, this time with the dtab table's vmap column. Compare the results of this output to those for the flex table. Note that maptostring returns the header_name parameter values you specified when you loaded the data:

```sql
=> SELECT MAPTOSTRING(vmap) FROM dtab;

maptostring

{ "CITY" : "CITY", "Name" : "Name", "New City" : "New city", "State" : "State", "Zip" : "zip" }

{ "CITY" : "BOSTON", "Name" : "Tom", "New City" : "boston", "State" : "MA", "Zip" : "02121" }

(4 rows)
```
7. Query the `delim` column to view the contents differently:

```sql
=> SELECT delim FROM dtab;
      delim
-------------------------------
 Name|CITY|New city|State|zip
-----|-----|--------|-----|-----
Tom  |BOSTON|boston   |MA   |02121
Eric |Burlington|BURLINGTON|MA |02482
Jamie|cambridge|CAMBRIDGE|MA |02811
(4 rows)
```

See Also

- [MAPJSONEXTRACTOR](#)
- [MAPREGEXEXTRACTOR](#)

**MAPITEMS**

Returns information about items in a VMap. Use this transform function with one or more optional arguments to access polystructured values within the VMap data. This function requires an `OVER()` clause.

**Syntax**

```sql
MAPITEMS(VMap_data [, passthrough_arg [,...]] )
```
Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
<tr>
<td>passthrough_</td>
<td>[Optional] One or more arguments indicating keys within the map data</td>
</tr>
<tr>
<td>arg</td>
<td>in VMap_data</td>
</tr>
</tbody>
</table>

Examples

The following examples illustrate using mapItems() with the over(PARTITION BEST) clause.

This example determines the number of virtual columns in the map data using a flex table, labeled darkmountain. Query using the count() function to return the number of virtual columns in the map data:

```
=> SELECT COUNT(keys) FROM (SELECT MAPITEMS(darkmountain.__raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;
  count
-----
   19
(1 row)
```

The next example determines what items exist in the map data:

```
=> SELECT * FROM (SELECT MAPITEMS(darkmountain.__raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;
   keys  |   values
----------|-----------
hike_safety | 50.6      
name       | Mt Washington
height     | 17000
hike_safety | 12.2      
name       | Denali
height     | 29029
hike_safety | 34.1      
name       | Everest
height     | 14000
hike_safety | 22.8      
name       | Kilimanjaro
type       | mountain
```
Directly Query a Key Value in a VMap

Review the following JSON input file, `simple.json`. In particular, notice the array called `three_Array`, and its four values:

```json
{
    "one": "one",
    "two": 2,
    "three_Array": [
        "three_One",
        "three_Two",
        3,
        "three_Four"
    ],
    "four": 4,
    "five_Map": {
        "five_One": 51,
        "five_Two": "Fifty-two",
        "five_Three": "fifty three",
        "five_Four": 54,
        "five_Five": "5 x 5"
    },
    "six": 6
}
```

1. Create a flex table, mapper:

```sql
=> CREATE FLEX TABLE mapper();
CREATE TABLE
```

1. Load `simple.json` into the flex table mapper:

```sql
=> COPY mapper FROM '/home/dbadmin/data/simple.json' parser fjsonparser (flatten_arrays=false, flatten_maps=false);
Rows Loaded
----------
1
(1 row)
```

2. Call `mapkeys` on the flex table's `__raw__` column to see the flex table's keys, but not the key submaps. The return values indicate `three_Array` as one of the virtual columns:

```sql
=> SELECT MAPKEYS(__raw__) OVER() FROM mapper;
keys
----------
```

Vertica Analytic Database (9.0.x)
3. Call `mapitems` on flex table mapper with `three_Array` as a pass-through argument to the function. The call returns these array values:

```
=> SELECT __identity__, mapitems(three_Array) OVER(PARTITION BY __identity__) FROM mapper;

<table>
<thead>
<tr>
<th><strong>identity</strong></th>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>three_One</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>three_Two</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>three_Four</td>
</tr>
</tbody>
</table>
```

(4 rows)

See Also

- `EMPTYMAP`
- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
- `MAPSIZE`
- `MAPTOSTRING`
- `MAPVALUES`
- `MAPVERSION`
MAPJSONEXTRACTOR

Extracts content of repeated JSON data objects, including nested maps, or data with an outer list of JSON elements. The USING PARAMETERS phrase specifies optional parameters for the function. Empty input does not generate a Warning or Error.

Note: The function fails if the output size of the function is greater than 65000.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_maps</td>
<td>BOOLEAN</td>
<td>[Optional] Flattens sub-maps within the JSON data, separating map levels with a period (.). Default value: true</td>
</tr>
<tr>
<td>flatten_arrays</td>
<td>BOOLEAN</td>
<td>[Optional] Converts lists to sub-maps with integer keys. Lists are not flattened by default. Default value: false</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>BOOLEAN</td>
<td>[Optional] Specifies whether to ignore duplicate records (false), or to reject duplicates (true). In either case, the load continues.  Default value: false</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>BOOLEAN</td>
<td>[Optional] Rejects any row containing a key without a value (reject_on_empty_key=true). Default value: false</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>BOOLEAN</td>
<td>[Optional] Omits any key from the load data that does not have a value (omit_empty_keys=true). Default value: false</td>
</tr>
<tr>
<td>start_point</td>
<td>CHAR</td>
<td>[Optional] Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the start_point value. The parser processes data after the first instance, and up to the second, ignoring any remaining data. Default value: none</td>
</tr>
</tbody>
</table>
Examples

These examples use the following sample JSON data:

```json
{  "id": "5001",  "type": "None" }
{  "id": "5002",  "type": "Glazed" }
{  "id": "5005",  "type": "Sugar" }
{  "id": "5007",  "type": "Powdered Sugar" }
{  "id": "5004",  "type": "Maple" }
```

Save this example data as `bake_single.json`, and load that file.

1. Create a flex table, `flexjson`:

```sql
=> CREATE FLEX TABLE flexjson();
CREATE TABLE
```

2. Use `COPY` to load the `bake_single.json` file with the `fjsonparser` parser:

```sql
=> COPY flexjson FROM '/home/dbadmin/data/bake_single.json' parser fjsonparser();
Rows Loaded
----------
  5
(1 row)
```

3. Create a columnar table, `coljson`, with an identity column (`id`), a `json` column, and a column to hold a VMap, called `vmap`:

```sql
=> CREATE TABLE coljson(id IDENTITY(1,1), json varchar(128), vmap long varbinary(10000));
CREATE TABLE
```

4. Use `COPY` to load the `bake_single.json` file into the `coljson` table, using the `mapjsonextractor` function:

```sql
=> COPY coljson (json, vmap AS MapJSONExtractor(json)) FROM '/home/dbadmin/data/bake_single.json';
Rows Loaded
----------
  5
(1 row)
```

5. Use the `map tostring` function for the flex table `flexjson` to output the `__raw__` column contents as strings:

```sql
=> SELECT MAPTOSTRING(__raw__) FROM flexjson limit 5;
map tostring
```
6. Use the maptostring function again, this time with the coljson table's vmap column and compare the results. The element order differs:

```sql
=> SELECT MAPTOSTRING(vmap) FROM coljson limit 5;
maptostring

{   
  "id" : "5001",
  "type" : "None"
}

{   
  "id" : "5002",
  "type" : "Glazed"
}

{   
  "id" : "5004",
  "type" : "Maple"
}

{   
  "id" : "5005",
  "type" : "Sugar"
}

{   
  "id" : "5007",
  "type" : "Powdered Sugar"
}
```

(5 rows)
MAPKEYS

Returns the virtual columns (and values) present in any VMap data. This transform function requires an `over(PARTITION BEST)` clause.

Syntax

```
MAPKEYS(VMap_data)
```

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The _raw_ column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

Examples

**Determine Number of Virtual Columns in Map Data**

This example shows how to create a query, using an `over(PARTITION BEST)` clause with a flex table, `darkdata` to find the number of virtual column in the map data. The table is populated with JSON tweet data.
Query Ordered List of All Virtual Columns in the Map

This example shows a snippet of the return data when you query an ordered list of all virtual columns in the map data:

```
=> SELECT * FROM (SELECT MAPKEYS(darkdata.__raw__) OVER(PARTITION BEST) FROM darkdata) AS a;
      count
-------
      550
(1 row)
```

```
=> SELECT * FROM (SELECT MAPKEYS(darkdata.__raw__) OVER(PARTITION BEST) FROM darkdata) AS a;

keys
-------

 contributors
coordinates
created_at
delete.status.id
delete.status.id_str
delete.status.user_id
delete.status.user_id_str
entities.hashtags
entities.media
entities.urls
entities.user_mentions
favorited
geo
d
..
..
user.statuses_count
user.time_zone
user.url
user.utc_offset
user.verified
(125 rows)
```

See Also

- EmptyMap
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
• MAPKEYSINFO
• MAPLOOKUP
• MAPSIZE
• MAPTOSTRING
• MAPVALUES
• MAPVERSION

MAPKEYSINFO

Returns virtual column information from a given map. This transform function requires an over(PARTITION BEST) clause.

Syntax

MAPKEYSINFO(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

Returns

This function is a superset of the MAPKEYS() function. It returns the following information about each virtual column:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>The virtual column names in the raw data.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>length</td>
<td>The data length of the key name, which can differ from the actual string length.</td>
</tr>
<tr>
<td>type_oid</td>
<td>The OID type into which the value should be converted. Currently, the type is always 116 for a LONG VARCHAR, or 199 for a nested map that is stored as a LONG VARBINARY.</td>
</tr>
<tr>
<td>row_num</td>
<td>The number of rows in which the key was found.</td>
</tr>
<tr>
<td>field_num</td>
<td>The field number in which the key exists.</td>
</tr>
</tbody>
</table>

Examples

This example shows a snippet of the return data you receive if you query an ordered list of all virtual columns in the map data:

```sql
=> SELECT * FROM (SELECT MAPKEYSINFO(darkdata.__raw__) OVER(PARTITION BEST) FROM darkdata) AS a;
```

<table>
<thead>
<tr>
<th>keys</th>
<th>length</th>
<th>type_oid</th>
<th>row_num</th>
<th>field_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>coordinates</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>created_at</td>
<td>30</td>
<td>116</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>entities.hashtags</td>
<td>93</td>
<td>199</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>entities.media</td>
<td>772</td>
<td>199</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>entities.urls</td>
<td>16</td>
<td>199</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>entities.user_mentions</td>
<td>16</td>
<td>199</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>favorited</td>
<td>1</td>
<td>116</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>geo</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>id</td>
<td>18</td>
<td>116</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>id_str</td>
<td>18</td>
<td>116</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>delete.status.id</td>
<td>18</td>
<td>116</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>delete.status.id_str</td>
<td>18</td>
<td>116</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>delete.status.user_id</td>
<td>9</td>
<td>116</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>delete.status.user_id_str</td>
<td>9</td>
<td>116</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>delete.status.id</td>
<td>18</td>
<td>116</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>delete.status.id_str</td>
<td>18</td>
<td>116</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>delete.status.user_id</td>
<td>9</td>
<td>116</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>delete.status.user_id_str</td>
<td>9</td>
<td>116</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>(550 rows)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also

- **EMPTYMAP**
- **MAPAGGREGATE**
MAPLOOKUP

Returns single-key values from VMAP data. This scalar function returns a LONG VARCHAR, with values, or NULL if the virtual column does not have a value.

Using maplookup is case insensitive to virtual column names. To avoid loading same-name values, set the fjsonparser parser reject_on_duplicate parameter to true when data loading.

You can control the behavior for non-scalar values in a VMAP (like arrays), when loading data with the fjsonparser or favroparser parsers and its flatten-arrays argument. See Loading JSON Data and the FJSONPARSER reference.

For information about using maplookup() to access nested JSON data, see Querying Nested Data.

Syntax

MAPLOOKUP(VMap_data, 'virtual_column_name' [USING PARAMETERS [case_sensitive={false | true}] [, buffer_size=n] ]

Parameters

| VMap_data     | Any VMap data. The VMap can exist as: |
- The __raw__ column of a flex table
- Data returned from a map function such as maplookup()
- Other database content

<table>
<thead>
<tr>
<th>virtual_column_name</th>
<th>The name of the virtual column whose values this function returns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size</td>
<td>[Optional parameter] Specifies the maximum length (in bytes) of each value returned for virtual_column_name. To return all values for virtual_column_name, specify a buffer_size equal to or greater than (=&gt;) the number of bytes for any returned value. Any returned values greater in length than buffer_size are rejected. <strong>Default value:</strong> 0 (No limit on buffer_size)</td>
</tr>
<tr>
<td>case_sensitive</td>
<td>[Optional parameter] Specifies whether to return values for virtual_column_name if keys with different cases exist. <strong>Example:</strong> (...) USING PARAMETERS case_sensitive=true) <strong>Default value:</strong> false</td>
</tr>
</tbody>
</table>

**Examples**

This example returns the values of one virtual column, user.location:

```sql
=> SELECT MAPLOOKUP(__raw__, 'user.location') FROM darkdata ORDER BY 1;
maplookup
----------
Chile
Nesnia
Uptown
.
chicago
(12 rows)
```

**Using maplookup buffer_size**

Use the buffer_size= parameter to indicate the maximum length of any value that maplookup returns for the virtual column you specify. If none of the returned key values can be greater than n bytes, use this parameter to allocate n bytes as the buffer_size.

For the next example, save this JSON data to a file, simple_name.json:
1. Create a flex table, logs.

2. Load the simple_name.json data into logs, using the fjsonparser. Specify the flatten_arrays option as True:

   ```sql
   => COPY logs FROM '/home/dbadmin/data/simple_name.json' 
      PARSER fjsonparser(flatten_arrays=True);
   ```

3. Use maplookup with buffer_size=0 for the logs table name key. This query returns all of the values:

   ```sql
   => SELECT MAPLOOKUP(__raw__, 'name' USING PARAMETERS buffer_size=0) FROM logs;
   MapLookup
   .......... 
   sierra
   ben
   janis
   jen
   (4 rows)
   ```

4. Next, call maplookup() three times, specifying the buffer_size parameter as 3, 5, and 6, respectively. Now, maplookup() returns values with a byte length less than or equal to (=) buffer_size:

   ```sql
   => SELECT MAPLOOKUP(__raw__, 'name' USING PARAMETERS buffer_size=3) FROM logs;
   MapLookup
   .......... 
   ```
Disambiguate Empty Output Rows

This example shows how to interpret empty rows. Using `maplookup` without first checking whether a key exists can be ambiguous. When you review the following output, 12 empty rows, you cannot determine whether a `user.location` key has:

- A non-NULL value
- A NULL value
- No value

```sql
=> SELECT MAPLOOKUP(__raw__, 'user.location') FROM darkdata;
maplookup
---------
(12 rows)
```
To disambiguate empty output rows, use the `mapcontainskey()` function in conjunction with `maplookup()`. When `maplookup` returns an empty field, the corresponding value from `mapcontainskey` indicates `t` for a NULL or other value, or `f` for no value.

The following example output using both functions lists rows with NULL or a name value as `t`, and rows with no value as `f`:

```
=> SELECT MAPLOOKUP(__raw__, 'user.location'), MAPCONTAINSKEY(__raw__, 'user.location')
FROM darkdata ORDER BY 1;
maplookup | mapcontainskey
-----------|---------------------------
        t |                t
        t |                t
        t |                t
        t |                t
Chile   |                t
Nesnia  |                t
Uptown  |                t
chicago |    f >>>>>>>>>No value
|    f >>>>>>>>>No value
|    f >>>>>>>>>No value
|    f >>>>>>>>>No value
(12 rows)
```

**Check for Case-Sensitive Virtual Columns**

You can use `maplookup()` with the `case_sensitive` parameter to return results when key names with different cases exist.

1. Save the following sample content as a JSON file. This example saves the file as `repeated_key_name.json`:

```json
{
    "test": "lower1"
}
{
    "TEST": "upper1"
}
{
    "TEst": "half1"
}
{
    "test": "lower2",
    "TEst": "half2"
}
{
    "TEST": "upper2",
    "TEst": "half3"
}
{
    "test": "lower3",
    "TEST": "upper3"
}
```
2. Create a flex table, dupe, and load the JSON file:

```sql
=> CREATE FLEX TABLE dupe();
CREATE TABLE
dbt=> COPY dupe FROM '/home/release/KData/repeated_key_name.json' parser fjsonparser();
Rows Loaded
-----------------
 8
(1 row)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPSIZE
- MAPTOSTRING
- MAPVALUES
- MAPVERSION
MAPPUT

Accepts a VMap and one or more key/value pairs and returns a new VMap with the key/value pairs added. Keys must be set using the auxiliary function SetMapKeys(), and can only be constant strings. If the VMap has any of the new input keys, then the original values are replaced by the new ones.

Syntax

MAPPUT(VMap_data, value1 [, value2, value3, ...] using parameters keys=SetMapKeys('key1'[, 'key2', 'key3', ...])

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMap_data</td>
<td>Any VMap data. The VMap can exist as:</td>
</tr>
<tr>
<td></td>
<td>- The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>- Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>- Other database content</td>
</tr>
<tr>
<td>value</td>
<td>One or more values to add to the VMap specified in VMap_data.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>The keys parameter must be the result of SetMapKeys(). SetMapKeys() simply takes one or more constant strings as arguments.</td>
</tr>
</tbody>
</table>

The following example shows how to create a flex table and use COPY to enter some basic JSON data. After creating a second flex table (vmapdata2), insert the new vmap results from mapput(), with additional key/value pairs.

1. Create sample table:

```sql
=> CREATE FLEX TABLE vmapdata1();
CREATE TABLE
```
2. Load sample JSON data from STDIN:

```sql
=> COPY vmapdata1 FROM stdin parser fjsonparser();
Enter data to be copied followed by a newline. 
End with a backslash and a period on a line by itself. 
>>{"aaa": 1, "bbb": 2, "ccc": 3} 
>> 
```

3. Create another flex table and use the function to insert data into it:

```sql
=> CREATE FLEX TABLE vmapdata2(); 
=> INSERT INTO vmapdata2 SELECT MAPPUT(_raw_, '7','8','9' 
using parameters keys=SetMapKeys('xxx','yyy','zzz')) from vmapdata1; 
```

4. View the difference between the original and the new flex tables:

```sql
=> SELECT MAPTOSTRING(_raw__) FROM vmapdata1; 
maptostring------------------------ 
| { 
| "aaa" : "1", 
| "bbb" : "2", 
| "ccc" : "3" 
| } 
| (1 row) 
=> SELECT MAPTOSTRING(_raw__) FROM vmapdata2; 
maptostring------------------------ 
| { 
| "mapput" : { 
| "aaa" : "1", 
| "bbb" : "2", 
| "ccc" : "3", 
| "xxx" : "7", 
| "yyy" : "8", 
| "zzz" : "9" 
| } 
| }
```

See Also

- **EMPTYMAP**
- **MAPAGGREGATE**
- **MAPCONTAINSKEY**
MAPREGEXEXTRACTOR

Extracts data from a regular expression and returns the results as a VMap. Use the USING PARAMETERS pattern= phrase, followed by the regular expression.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern=</td>
<td>VARCHAR</td>
<td>The regular expression as a string. Default value: An empty string (&quot; &quot;).</td>
</tr>
<tr>
<td>record_terminator</td>
<td>VARCHAR</td>
<td>[Optional] The character used to separate input records. Default value: \n.</td>
</tr>
<tr>
<td>logline_column</td>
<td>VARCHAR</td>
<td>[Optional] The destination column containing the full string that the regular expression matched. Default value: An empty string (&quot; &quot;).</td>
</tr>
</tbody>
</table>
Examples

These examples use the following regular expression, which searches for information that includes the timestamp, date, thread_name, and thread_id strings.

Caution: For display purposes, this sample regular expression adds new line characters to split long lines of text. To use this expression in a query, first copy and edit the example to remove any new line characters.

This example expression loads any thread_id hex value, regardless of whether it has a 0x prefix, ((thread_id(?:0x)?[0-9a-f]+).

'^(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d.\d+) (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+) -(?(?<transaction_id>[0-9a-f])?):[\[]<component>\w+\][\[]\w+\]<\(\w+\) @?\(?<enode>\w+\): (?<text>.*'))

The following examples may include newline characters for display purposes.

1. Create a flex table, flogs:

   => CREATE FLEX TABLE flogs();

2. Use COPY to load a sample log file (vertica.log), using the flex table fregexparsel. Note that this example includes added line characters for displaying long text lines.

   => COPY flogs FROM '/home/dbadmin/tempdat/vertica.log' PARSER FREGEXPARSER(pattern='^\(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d.\d+) (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+) -(?(?<transaction_id>[0-9a-f])?):[\[]<component>\w+\][\[]\w+\]<\(\w+\) @?\(?<enode>\w+\): (?<text>.*'));

   Rows Loaded
   ------------
   81399
   (1 row)

3. Use MapToString to return the results from calling MapRegexExtractor with a regular expression. The output returns the results of the function in string format.

   => SELECT MAPTOSTRING(MapregexExtractor(E'2014-04-02 04:02:51.011 TM Moveout:0x2aab08008f860-a0000000002067 [Txn] <INFO> Begin Txn: a0000000002067 'Moveout: Tuple Mover' using PARAMETERS pattern='(\(?<time>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d.\d+) (?<thread_name>[A-Za-z ]+):(?<thread_id>(?:0x)?[0-9a-f]+) -(?(?<transaction_id>[0-9a-f])?):[\[]<component>\w+\][\[]\w+\]<\(\w+\) @?\(?<enode>\w+\): (?<text>.*')) FROM flogs where _identity_=13;
See Also

- MAPDELIMITEDEXTRACTOR
- MAPJSONEXTRACTOR

MAPSIZE

Returns the number of virtual columns present in any VMap data. Use this scalar function to determine the size of keys.

Syntax

MAPSIZE(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The <code>__raw__</code> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>Other database content</td>
</tr>
</tbody>
</table>

```sql
maptostring
-- 
{
  "component": "Txn",
  "level": "INFO",
  "text": "Begin Txn: a0000000002067 'Moveout: Tuple Mover'",
  "thread_id": "0x2aab9000f860",
  "thread_name": "TM Moveout",
  "time": "2014-04-02 04:02:51.011",
  "transaction_id": "a0000000002067"
}
(1 row)
```
Examples

This example shows the returned sizes from the number of keys in the flex table darkmountain:

```sql
=> SELECT MAPSIZE(__raw__) FROM darkmountain;
mapsize
--------
3
4
4
4
4
(5 rows)
```

See Also

- `EMPTYMAP`
- `MAPAGGREGATE`
- `MAPCONTAINSKEY`
- `MAPCONTAINSVALUE`
- `MAPITEMS`
- `MAPKEYS`
- `MAPKEYSINFO`
- `MAPLOOKUP`
- `MAPTOSTRING`
- `MAPVALUES`
- `MAPVERSION`

MAPTOSTRING

Recursively builds a string representation VMap data, including nested JSON maps. Use this transform function to display the VMap contents in a readable LONG VARCHAR format. Use mptostring to see how map data is nested before querying virtual columns with mapvalues().
Syntax

MAPTOSTRING(VMap_data [using parameters canonical_json={true | false}])

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>Any VMap data. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>canonical_json</th>
<th>=bool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Optional parameter]</td>
</tr>
<tr>
<td></td>
<td>Produces canonical JSON output by default, using the first instance of any duplicate keys in the map data.</td>
</tr>
<tr>
<td></td>
<td>Use this parameter as other UDF parameters, preceded by using parameters, as shown in the examples. Setting this argument to false maintains the previous behavior of mptostring() and returns same-name keys and their values.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> canonical-json=true</td>
</tr>
</tbody>
</table>

Examples

The following example shows how to create a sample flex table, darkdata and load JSON data from STDIN. By calling mptostring() twice with both values for the canonical_json parameter, you can see the different results on the flex table __raw__ column data.
1. Create sample table:

```sql
=> CREATE FLEX TABLE darkdata();
CREATE TABLE
```

2. Load sample JSON data from STDIN:

```sql
=> COPY darkdata FROM stdin parser fjsonparser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> {"aaa": 1, "aaa": 2, "AAA": 3, "bbb": "aaa\bbb"
>> \.
```

3. Call mptostring() with its default behavior using canonical JSON output, and then review the flex table contents. The function returns the first duplicate key and its value ("aaa": "1") but omits remaining duplicate keys ("aaa": "2"):

```sql
=> SELECT MAPTOSTRING(__raw__) FROM darkdata;
maptostring
-----------------------------------------------
{  
    "AAA": "3",
    "aaa": "1",
    "bbb": "aaa\bbb"
}
(1 row)
```

4. Next, call mptostring() with using parameters canonical_json=false). This time, the function returns the first duplicate keys and their values:

```sql
=> SELECT MAPTOSTRING(__raw__ using parameters canonical_json=false) FROM darkdata;
maptostring
-----------------------------------------------
{
    "aaa": "1",
    "aaa": "2",
    "AAA": "3",
    "bbb": "aaa\bbb"
}
(1 row)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
• MAPCONTAINSKEY
• MAPCONTAINSVALUE
• MAPITEMS
• MAPKEYS
• MAPKEYSINFO
• MAPLOOKUP
• MAPSIZE
• MAPVALUES
• MAPVERSION

MAPVALUES

Returns a string representation of the top-level values from a VMap. This transform function requires an over() clause.

Syntax

MAPVALUES(VMap_data)

Arguments

<table>
<thead>
<tr>
<th>VMap_data</th>
<th>The VMap from which values should be returned. The VMap can exist as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The <strong>raw</strong> column of a flex table</td>
</tr>
<tr>
<td></td>
<td>• Data returned from a map function such as maplookup()</td>
</tr>
<tr>
<td></td>
<td>• Other database content</td>
</tr>
</tbody>
</table>

Examples

The following example shows how to query a darkmountain flex table, using an over() clause (in this case, the over(PARTITION BEST) clause) with mapvalues().
=> SELECT * FROM (SELECT MAPVALUES(darkmountain._raw__) OVER(PARTITION BEST) FROM darkmountain) AS a;
values
-------------------
29029
34.1
Everest
mountain
29029
15.4
Mt St Helens
volcano
17000
12.2
Denali
mountain
14000
22.8
Kilimanjaro
mountain
50.6
Mt Washington
mountain
(19 rows)

See Also

- EMPTYPARTKEY
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPSTRING
- MAPVERSION
MAPVERSION

Returns the version or invalidity of any map data. This scalar function returns the map version (such as 1) or -1, if the map data is invalid.

Syntax

MAPVERSION(VMap_data)

Arguments

| VMap_data | The VMap data either from a __raw__ column in a flex table or from the data returned from a map function such as maplookup(). |

Examples

The following example shows how to use mapversion() with the darkmountainflex table, returning mapversion 1 for the flex table map data:

```sql
=> SELECT MAPVERSION(__raw__) FROM darkmountain;
  version
------------
       1
       1
       1
       1
       1
(5 rows)
```

See Also

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
**MATERIALIZE_FLEXTABLE_COLUMNS**

Materializes virtual columns listed as `key_names` in the `flextable_keys` table you compute using either `COMPUTE_FLEXTABLE_KEYS` or `COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW`.

**Note:** Each column that you materialize with this function counts against the data storage limit of your license. To check your Vertica license compliance, call the `AUDIT()` or `AUDIT_FLEX()` functions.

**Syntax**

```
MATERIALIZE_FLEXTABLE_COLUMNS('[[[database.]schema.]flextable] [, n-columns [, keys-table-name]]
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td><code>flextable</code></td>
<td>The name of the flex table with columns to materialize. Specifying only the flex table name attempts to materialize up to 50 columns of key names in the default <code>flextable_keys</code> table. When you use this argument, the function:</td>
</tr>
<tr>
<td></td>
<td>- Skips any columns already materialized</td>
</tr>
</tbody>
</table>
Ignores any empty keys

To materialize a specific number of columns, use the optional parameter n_columns, described next.

**n-columns**

The number of columns to materialize. The function attempts to materialize the number of columns from the flex_table_keys table, skipping any columns already materialized.

Vertica tables support a total of 1600 columns, which is the largest value you can specify for n-columns. The function orders the materialized results by frequency, descending, key_name when materializing the first n columns.

**keys-table-name**

The name of a flex_keys_table from which to materialize columns. The function:

- Materializes the number of columns (value of n-columns) from keys-table-name
- Skips any columns already materialized
- Orders the materialized results by frequency, descending, key_name when materializing the first n columns.

Examples

The following example shows how to call MATERIALIZE_FLEXTABLE_COLUMNS to materialize columns. First, load a sample file of tweets (tweets_10000.json) into the flex table twitter_r.

After loading data and computing keys for the sample flex table, call materialize_flextable_columns to materialize the first four columns:

```sql
=> COPY twitter_r FROM '/home/release/KData/tweets_10000.json' parser fjsonparser();
Rows Loaded
-------------
10000
(1 row)

=> SELECT compute_flextable_keys ('twitter_r');
compute_flextable_keys
-------------------------------------------------------------------------------
Please see public.twitter_r_keys for updated keys
(1 row)

=> select materialize_flextable_columns('twitter_r', 4);
```
materialize_flextable_columns

The following columns were added to the table public.twitter_r:
  contributors
  entities.hashtags
  entities.urls

For more details, run the following query:
SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and table_name = 'twitter_r';

(1 row)

The last message in the example recommends querying system table MATERIALIZE_FLEXTABLE_COLUMNS_RESULTS for the results of materializing the columns, as shown:

```
=> SELECT * FROM v_catalog.materialize_flextable_columns_results WHERE table_schema = 'public' and table_name = 'twitter_r';
  table_id | table_schema | table_name | creation_time | key_name
-----------------+-----------------+------------+---------------+----------
      4503599627373 | public | twitter_r | 2013-11-20 17:00:27.945484-05 | contributors
ADDED | Added successfully
      4503599627373 | public | twitter_r | 2013-11-20 17:00:27.94551-05 | entities.hashtags
ADDED | Added successfully
      4503599627373 | public | twitter_r | 2013-11-20 17:00:27.945519-05 | entities.urls
ADDED | Added successfully
      4503599627373 | public | twitter_r | 2013-11-20 17:00:27.945532-05 | created_at
EXISTS | Column of same name already added
```

See Also

- BUILD_FLEXTABLE_VIEW
- COMPUTE_FLEXTABLE_KEYS
- COMPUTE_FLEXTABLE_KEYS_AND_BUILD_VIEW
- RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW

Restores the _keys table and the _view. The function also links the _keys table with its associated flex table, in cases where either table is dropped. The function also indicates whether it restored one or both objects.
Syntax

RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('flex_table')

Arguments

| flex_table | The name of a flex table. |

Examples

This example shows how to invoke this function with an existing flex table, restoring both the _keys table and _view:

=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
   RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
-----------------------------
The keys table public.darkdata_keys was restored successfully.
The view public.darkdata_view was restored successfully.
(1 row)

This example illustrates that the function restored darkdata_view, but that darkdata_keys did not need restoring:

=> SELECT RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW('darkdata');
   RESTORE_FLEXTABLE_DEFAULT_KEYS_TABLE_AND_VIEW
-----------------------------
The keys table public.darkdata_keys already exists and is linked to darkdata.
The view public.darkdata_view was restored successfully.
(1 row)

After restoring the _keys table, there is no content. To populate the flex keys, call the COMPUTE_FLEXTABLE_KEYS meta function.

=> SELECT * FROM darkdata_keys;
   key_name | frequency | data_type_guess
----------|-----------|-----------------|
(0 rows)

See Also

- BUILD_FLEXTABLE_VIEW
- COMPUTE_FLEXTABLE_KEYS
Hadoop Functions

This section contains functions to manage interactions with Hadoop.

CLEAR_HDFS_CACHES

Clears the configuration information copied from HDFS and any cached connections.

This function affects reads using the hdfs scheme in the following ways:

- This function flushes information loaded from configuration files copied from Hadoop (such as core-site.xml). These files are found on the path set by the HadoopConfDir configuration parameter.

- This function flushes information about which Name Node is active in a High Availability (HA) Hadoop cluster. Therefore, the first request to Hadoop after calling this function is slower than expected.

Vertica maintains a cache of open connections to name nodes to reduce latency. This function flushes that cache.

Syntax

CLEAR_HDFS_CACHES ( )

Privileges

Superuser

Example

The following example clears the Hadoop configuration information:

```sql
=> SELECT CLEAR_HDFS_CACHES();
CLEARED_HDFS_CACHES
--------------------
Cleared
```
See Also

Apache Hadoop Parameters

**KERBEROS_HDFS_CONFIG_CHECK**

Tests the Kerberos configuration of a Vertica cluster that uses HDFS. This function is a more specific version of **KERBEROS_CONFIG_CHECK**.

You can call this function with arguments to specify an HDFS configuration to test, or without arguments. If you call it with no arguments, this function reads the HDFS configuration files and fails if it does not find them. See Configuring the hdfs Scheme. If it finds configuration files, it tests all configured nameservices.

The function performs the following tests, in order:

- Are Kerberos services available?
- Does a keytab file exist and are the Kerberos and HDFS configuration parameters set in the database?
- Can Vertica read and invoke kinit with the keys to authenticate to HDFS and obtain the database Kerberos ticket?
- Can Vertica perform hdfs and webhdfs operations using both the database Kerberos ticket and user-forwardable tickets?
- Can Vertica make unauthenticated WebHCat calls?

If any test fails, the function returns a descriptive error message.

**Syntax**

```
KERBEROS_HDFS_CONFIG_CHECK( ['hdfsHost:hdfsPort',
   'webhdfsHost:webhdfsPort', 'webhcatHost:webhcatPort'] )
```

**Arguments**

| hdfsHost, hdfsPort | The hostname or IP address and port of the HDFS name node. Vertica uses this server to access data that is specified with hdfs |
URLs. If the value is '', the function skips this part of the check.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>webhdfsHost</code>, <code>webhdfsPort</code></td>
<td>The hostname or IP address and port of the WebHDFS server. Vertica uses this server to access data that is specified with webhdfs URLs. If the value is '', the function skips this part of the check.</td>
</tr>
<tr>
<td><code>webhcatHost</code>, <code>webhcatPort</code></td>
<td>The hostname or IP address and port of the WebHCat server. The HCatalog Connector uses this server to access data through Hive. If the value is '', the function skips this part of the check.</td>
</tr>
</tbody>
</table>

Privileges

This function does not require privileges.

Examples

The following example shows the results when the function infers the configuration from the HDFS configuration files. In this example, the configuration files use HiveServer2, not WebHCat, so the function skips the WebHCat part of the test.

```sql
=> SELECT KERBEROS_HDFS_CONFIG_CHECK();
kerberos_hdfs_config_check
---------------------------------------------
-------- Checking Hadoop Kerberos configuration--------

HadoopFS parameters:
  Principal: [vertica/data.hadoop.com@EXAMPLE.COM]
  Token Refresh Frequency: 0
  Connection Timeout: 60
  Authentication (default): [/scratch_b/user/data/vertica.user.q18v05.keytab]
ok: krb5 exists at [/scratch_b/user/data/q18v05.krb5.conf]
ok: Vertica can kinit using keytab file
ok: User [user_1] has valid client authentication for kerberos principal [user_1@EXAMPLE.COM]

**Inferring HDFS configuration**
  Number of HDFS Clusters: 1
  Cluster 1: Nameservice [ns-q18v05] has [2] Namenodes [q18v05.example.com:8020]
  [q18v06.example.com:8020] and has [High Availability][No RPC Encryption][Kerberos Authentication][HTTP Enabled]

---------- Checking LibHdfs++ ----------

Attempting to check default nameservice [hdfs://] via Libhdfs++
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

Attempting to check cluster [ns-q18v05] via Libhdfs++
```
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

---------- Checking WebHdfs ----------

Attempting to check namenode URL [http://q18v05.example.com:50070] via WebHdfs
ok: Can make unauthenticated external curl connection
ok: Can make authenticated external curl connection
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

Attempting to check namenode URL [http://q18v06.example.com:50070] via WebHdfs
ok: Can make unauthenticated external curl connection
Notice: In standby mode

---------- Checking HCatalog ----------
Skipping WebHdfs tests since no valid servers were found
(1 row)

The following example uses parameters to specify the HDFS and WebHDFS hosts to check, omitting WebHCat.

=> SELECT KERBEROS_HDFS_CONFIG_CHECK('q18v05.example.com:8020',
   'http://q18v05.example.com:50070', '');
kerberos_hdfs_config_check

---------- Checking Hadoop Kerberos configuration ----------

HadoopFS parameters:
  Principal: [vertica/data.hadoop.com@EXAMPLE.COM]
  Token Refresh Frequency: 0
  Connection Timeout: 0
  Authentication (default): [/scratch_b/user/data/vertica.user.q18v05.keytab]
ok: krb5 exists at [/scratch_b/user/data/q18v05.krb5.conf]
ok: Vertica can kinit using keytab file
ok: User [user_1] has valid client authentication for kerberos principal [user_1@EXAMPLE.COM]

---------- Checking LibHdfs++ ----------

Attempting to check default nameservice [hdfs://] via LibHdfs++
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

Attempting to check cluster [q18v05.example.com:8020] via LibHdfs++
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

---------- Checking WebHdfs ----------

Attempting to check namenode URL [http://q18v05.example.com:50070] via WebHdfs
ok: Can make unauthenticated external curl connection
ok: Can make authenticated external curl connection
ok: Can access using ticket for [Vertica]
ok: Can access using ticket for [user_1]

---------- Checking HCatalog ----------
SYNC_WITH_HCATALOG_SCHEMA

Copies the structure of a Hive database schema available through the HCatalog Connector to a Vertica schema.

Syntax

SYNC_WITH_HCATALOG_SCHEMA( local_schema, hcatalog_schema, [drop_tables] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_schema</td>
<td>The existing Vertica schema to store the copied HCatalog schema's metadata. This can be the same schema as hcatalog_schema, or it can be a separate one created using CREATE SCHEMA.</td>
</tr>
<tr>
<td>hcatalog_schema</td>
<td>The HCatalog schema to copy. This is the schema created using CREATE HCATALOG SCHEMA.</td>
</tr>
<tr>
<td>[drop_tables]</td>
<td>If true, drop any tables in local_schema that do not correspond to a table in hcatalog_schema</td>
</tr>
</tbody>
</table>

Notes

The SYNC_WITH_HCATALOG_SCHEMA function overwrites tables in the Vertica schema whose names match a table in the HCatalog schema. Do not use the Vertica schema to store other data.

Hive STRING and BINARY data types are matched, in Vertica, to VARCHAR(65000) and VARBINARY(65000) types. You can use ALTER TABLE to adjust these after creating the schema. The maximum size of a VARCHAR or VARBINARY in Vertica is 65000, but you can use LONG VARCHAR and LONG VARBINARY to specify larger values.

Hive and Vertica define string length in different ways. In Hive the length is the number of characters; in Vertica it is the number of bytes. Thus, a character encoding that uses more than one byte, such as Unicode, can result in mismatches between the two. Set values in Vertica based on bytes, not characters, to avoid data truncation.
If the data size exceeds the declared size for the column, Vertica logs an event at read time in the QUERYEVENTS system table.

This function can synchronize the HCatalog schema directly, in which case you call it with the same schema name for both parameters. It can also synchronize a different schema to the HCatalog schema.

If you change the settings of any HCatalog Connector configuration parameters (Apache Hadoop Parameters), you must call this function again.

Privileges

The user must have CREATE privileges on Local_schema.

The user also requires access to Hive data in one of the following ways:

- Have USAGE permissions on hcatalog_schema, if Hive does not use an authorization service to manage access.
- Have permission through an authorization service (Sentry or Ranger), and have access to the underlying files in HDFS. (Sentry can provide that access through ACL synchronization.)
- Be the dbadmin user, with or without an authorization service.

Examples

The following example shows using SYNC_WITH_HCATALOG_SCHEMA to synchronize an HCatalog schema named hcat:

```sql
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost' HCATALOG_SCHEMA='default'
    HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');
sync_with_hcatalog_schema
---------------------------------------------------------------
Schema hcat synchronized with hcat
tables in hcat = 56
tables altered in hcat = 0
tables created in hcat = 56
stale tables in hcat = 0
table changes erred in hcat = 0
(1 row)
=> -- Use vsql's \d command to describe a table in the synced schema
=> \d hcat.messages
List of Fields by Tables
| Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign Key |
```

Vertica Documentation
SQL Reference Manual
Vertica Analytic Database (9.0.x)
The following example shows using SYNC_WITH_HCATALOG_SCHEMA followed by ALTER TABLE to adjust a column value:

```sql
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost' HCATALOG_SCHEMA='default'
  -> HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');
...
=> ALTER TABLE hcat.t ALTER COLUMN a1 SET DATA TYPE long varchar(1000000);
=> ALTER TABLE hcat.t ALTER COLUMN a2 SET DATA TYPE long varbinary(1000000);
```

The following example shows using SYNC_WITH_HCATALOG_SCHEMA with a local (non-HCatalog) schema:

```sql
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost' HCATALOG_SCHEMA='default'
  -> HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> CREATE SCHEMA hcat_local;
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat_local', 'hcat');
```

**SYNC_WITH_HCATALOG_SCHEMA_TABLE**

Copies the structure of a single table in a Hive database schema available through the HCatalog Connector to a Vertica table.

**Syntax**

```
SYNC_WITH_HCATALOG_SCHEMA_TABLE( local_schema, hcatalog_schema, table_name )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local_schema</strong></td>
<td>The existing Vertica schema to store the copied HCatalog schema's metadata. This can be the same schema as <strong>hcatalog_schema</strong>, or it can be a separate one created using CREATE SCHEMA.</td>
</tr>
<tr>
<td><strong>hcatalog_schema</strong></td>
<td>The HCatalog schema to copy. This is the schema created using CREATE HCATALOG SCHEMA.</td>
</tr>
</tbody>
</table>
**table_name** The table in **hcatalog_schema** to copy.

---

**Notes**

If **table_name** does not exist in **hcatalog_schema**, this function returns an error.

If **table_name** already exists in **local_schema**, the **SYNC_WITH_HCATALOG_SCHEMA_TABLE** function overwrites it.

Hive STRING and BINARY data types are matched, in Vertica, to VARCHAR(65000) and VARBINARY(65000) types. You can use **ALTER TABLE** to adjust these after creating the schema. The maximum size of a VARCHAR or VARBINARY in Vertica is 65000, but you can use LONG VARCHAR and LONG VARBINARY to specify larger values.

Hive and Vertica define string length in different ways. In Hive the length is the number of characters; in Vertica it is the number of bytes. Thus, a character encoding that uses more than one byte, such as Unicode, can result in mismatches between the two. Set values in Vertica based on bytes, not characters, to avoid data truncation.

If the data size exceeds the declared size for the column, Vertica logs an event at read time in the **QUERY_EVENTS** system table.

This function can synchronize the HCatalog schema directly, in which case you call it with the same schema name for both of the **local_schema** and **hcatalog_schema** parameters. It can also synchronize a different schema to the HCatalog schema.

---

**Privileges**

The user must have CREATE privileges on **local_schema**.

The user also requires access to Hive data in one of the following ways:

- Have USAGE permissions on **hcatalog_schema**, if Hive does not use an authorization service to manage access.

- Have permission through an authorization service (Sentry or Ranger), and have access to the underlying files in HDFS. (Sentry can provide that access through ACL synchronization.)

- Be the dbadmin user, with or without an authorization service.

---

**Examples**

The following example shows using **SYNC_WITH_HCATALOG_SCHEMA_TABLE** to synchronize the "nation" table:
```sql
=> CREATE SCHEMA 'hcat_local';
CREATE SCHEMA

=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost' HCATALOG_SCHEMA='hcat'
    HCATALOG_USER='hcatuser';
CREATE SCHEMA

=> SELECT sync_with_hcatalog_schema_table('hcat_local', 'hcat', 'nation');
sync_with_hcatalog_schema_table
-------------------------------
    Schema hcat_local synchronized with hcat for table nation
    table nation is created in schema hcat_local
    (1 row)
```

The following example shows the behavior if the "nation" table already exists in the local schema:

```sql
=> SELECT sync_with_hcatalog_schema_table('hcat_local','hcat','nation');
sync_with_hcatalog_schema_table
-------------------------------
    Schema hcat_local synchronized with hcat for table nation
    table nation is altered in schema hcat_local
    (1 row)
```

**VERIFY_HADOOP_CONF_DIR**

Verifies that the Hadoop configuration that is used to access HDFS is valid on all Vertica nodes. The configuration is valid if:

- all required configuration files are found on the path defined by the HadoopConfDir configuration parameter
- all properties needed by Vertica are set in those files

This function does not attempt to validate the settings of those properties; it only verifies that they have values.

It is possible for Hadoop configuration to be valid on some nodes and invalid on others. The function reports a validation failure if the value is invalid on any node; the rest of the output reports the details.

**Syntax**

```sql
VERIFY_HADOOP_CONF_DIR( )
```
Parameters

This function has no parameters.

Privileges

This function does not require privileges.

Examples

The following example shows the results when the Hadoop configuration is valid.

```sql
=> SELECT VERIFY_HADOOP_CONF_DIR();
verify_hadoop_conf_dir

Validation Success
v_vmart_node0001: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0002: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0003: HadoopConfDir [PG_TESTOUT/config] is valid
v_vmart_node0004: HadoopConfDir [PG_TESTOUT/config] is valid
(1 row)
```

In the following example, the Hadoop configuration is valid on one node, but on other nodes a needed value is missing.

```sql
=> SELECT VERIFY_HADOOP_CONF_DIR();
verify_hadoop_conf_dir

Validation Failure
v_vmart_node0001: HadoopConfDir [PG_TESTOUT/test_configs/config] is valid
v_vmart_node0002: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
v_vmart_node0003: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
v_vmart_node0004: No fs.defaultFS parameter found in config files in [PG_TESTOUT/config]
(1 row)
```

LDAP Link Functions

This section contains the functions associated with the Vertica LDAP Link function.

In This Section
LDAP_LINK_SYNC_START

Begins the synchronization between the LDAP server and Vertica immediately rather than waiting for the interval set in LDAPLinkInterval.

Syntax

ldap_link_sync_start()

Privileges

You must be a dbadmin user.

Example

=> SELECT ldap_link_sync_start();

See Also

LDAP Link Parameters
License Management Functions

This section contains function that monitor Vertica license status and compliance.

AUDIT

Returns the raw data size (in bytes) of a database, schema, or table as it is counted in an audit of the database size. Unless you specify zero error tolerance and 100 percent confidence level, AUDIT returns only approximate results that can vary over multiple iterations.

AUDIT estimates the size using the same data sampling method as Vertica uses, to determine if a database complies with the licensed database size allowance. Vertica does not use these results to determine whether the size of the database complies with the Vertica license's data allowance. For details, see Auditing Database Size in the Administrator's Guide.

Syntax

AUDIT('[[[database.]schema.]scope ][, 'granularity'[, error-tolerance[, confidence-level]]] )

Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>myschema.myObject</td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>scope</th>
<th>Specifies the extent of the audit:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Empty string ('') audits the entire database.</td>
</tr>
<tr>
<td></td>
<td>The name of the schema or table to audit.</td>
</tr>
<tr>
<td></td>
<td>The schema or table to audit. To audit the database, set this parameter to an empty string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>granularity</th>
<th>The level at which the audit reports its results, one of the following strings:</th>
</tr>
</thead>
</table>
*database*  
*schema*  
*table*

The level of granularity must be equal to or less than the granularity of *scope*. If you omit this parameter, granularity is set to the same level as *scope*. Thus, if `online_sales` is a schema, the following statements are identical:

```sql
AUDIT('online_sales', 'schema');
AUDIT('online_sales');
```

If `AUDIT` sets granularity to a level lower than the target object, it returns with a message that refers you to system table `USER_AUDITS`. For details, see Querying `V_CATALOG.USER_AUDITS`, below.

**error-tolerance**

Specifies the percentage margin of error allowed in the audit estimate. Enter the tolerance value as a decimal number, between 0 and 100. The default value is 5, for a 5% margin of error.

Setting this value to 0 results in a full database audit, which is very resource intensive, as `AUDIT` analyzes the entire database. A full database audit significantly impacts performance, so Vertica does not recommend it for a production database.

Caution: Due to the iterative sampling that the auditing process uses, setting the error tolerance to a small fraction of a percent (for example, 0.00001) can cause `AUDIT` to run for a longer period than a full database audit. The lower you specify this value, the more resources the audit uses, as it performs more data sampling.

**confidence-level**

Specifies the statistical confidence level percentage of the estimate. Enter the confidence value as a decimal number, between 0 and 100. The default value is 99, indicating a confidence level of 99%.

The higher the confidence value, the more resources the function uses, as it performs more data sampling. Setting this value to 100 results in a full audit of the database, which is very resource intensive, as the function analyzes all of the database. A full
database audit significantly impacts performance, so Vertica does not recommend it for a production database.

Privileges

Superuser, or one of the following:

- SELECT privilege on the target tables
- USAGE privilege on the target schemas

**Note:** If you audit a schema or the database, Vertica only returns the size of all objects that you have privileges to access within the audited object, as described above.

Querying V_CATALOG.USER_AUDITS

If AUDIT sets granularity to a level lower than the target object, it returns with a message that refers you to system table USER_AUDITS. To obtain audit data on objects of the specified granularity, query this table. For example, the following query seeks to audit all tables in the store schema:

```sql
=> SELECT AUDIT('store', 'table');
AUDIT
See table sizes in v_catalog.user_audits for schema store
(1 row)
```

The next query queries USER_AUDITS and obtains the latest audits on those tables:

```sql
=> SELECT object_name, AVG(size_bytes)::int size_bytes, MAX(audit_start_timestamp::date) audit_start
    FROM user_audits WHERE object_schema='store'
    GROUP BY rollup(object_name) HAVING GROUPING_ID(object_name) < 1 ORDER BY GROUPING_ID();
```

<table>
<thead>
<tr>
<th>object_name</th>
<th>size_bytes</th>
<th>audit_start</th>
</tr>
</thead>
<tbody>
<tr>
<td>store_dimension</td>
<td>22067</td>
<td>2017-10-26</td>
</tr>
<tr>
<td>store_orders_fact</td>
<td>27201312</td>
<td>2017-10-26</td>
</tr>
<tr>
<td>store_sales_fact</td>
<td>301260170</td>
<td>2017-10-26</td>
</tr>
</tbody>
</table>

(3 rows)

Examples

See [Auditing Database Size](#).
AUDIT_FLEX

Returns the estimated ROS size of __raw__ columns, equivalent to the export size of the flex data in the audited objects. You can audit all flex data in the database, or narrow the audit scope to a specific flex table, projection, or schema. Vertica stores the audit results in system table USER_AUDITS.

The audit excludes the following:

- Flex keys
- Other columns in the audited tables.
- Temporary flex tables

Syntax

AUDIT_FLEX ('[scope]')

Parameters

<table>
<thead>
<tr>
<th>scope</th>
<th>Specifies the extent of the audit:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Empty string (' ') audits all flexible tables in the database.</td>
</tr>
<tr>
<td></td>
<td>The name of a schema, projection, or flex table.</td>
</tr>
</tbody>
</table>

Privileges

Superuser, or one of the following:

- SELECT privilege on the target tables
- USAGE privilege on the target schemas

Note: If you audit a schema or the database, Vertica only returns the size of all objects that you have privileges to access within the audited object, as described above.

Examples

Audit all flex tables in the current database:
dbs=> select audit_flex('');
  audit_flex
--------------
  8567679
(1 row)

Audit the flex tables in schema public:

dbs=> select audit_flex('public');
  audit_flex
--------------
  8567679
(1 row)

Audit the flex data in projection bakery_b0:

dbs=> select audit_flex('bakery_b0');
  audit_flex
--------------
  8566723
(1 row)

Audit flex table bakery:

dbs=> select audit_flex('bakery');
  audit_flex
--------------
  8566723
(1 row)

To report the results of all audits saved in the USER_AUDITS, the following shows part of an extended display from the system table showing an audit run on a schema called test, and the entire database, dbs:

dbs=> \x
Expanded display is on.

dbs=> select * from user_audits;
- RECORD 1 ------------------------------------------+
size_bytes | 0
user_id | 45035996273704962
user_name | release
object_id | 45035996273736664
object_type | SCHEMA
object_schema | 
object_name | test
audit_start_timestamp | 2014-02-04 14:52:15.126592-05
audit_end_timestamp | 2014-02-04 14:52:15.139475-05
confidence_level_percent | 99
error_tolerance_percent | 5
used_sampling | f
confidence_interval_lower_bound_bytes | 0
confidence_interval_upper_bound_bytes | 0
sample_count | 0
### AUDIT_LICENSE_SIZE

Triggers an immediate audit of the database size to determine if it is in compliance with the raw data storage allowance included in your Vertica licenses.

**Syntax**

```
AUDIT_LICENSE_SIZE()
```

**Privileges**

Superuser

**Example**

```
=> SELECT audit_license_size();
audit_license_size
---------------------
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization : 0%
Audit Time : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```
AUDIT_LICENSE_TERM

Triggers an immediate audit to determine if the Vertica license has expired.

Syntax

AUDIT_LICENSE_TERM()

Privileges

Superuser

Example

```sql
=> SELECT audit_license_term();
  audit_license_term
----------------------
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization : 0%
Audit Time : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```

DISPLAY_LICENSE

Returns the terms of your Vertica license. The information this function displays is:

- The start and end dates for which the license is valid (or "Perpetual" if the license has no expiration).
- The number of days you are allowed to use Vertica after your license term expires (the grace period)
- The amount of data your database can store, if your license includes a data allowance.

Syntax

DISPLAY_LICENSE()
Privileges
None

Examples

```sql
=> SELECT DISPLAY_LICENSE();
    DISPLAY_LICENSE
---------
Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB
(1 row)
```

**GET_AUDIT_TIME**

Reports the time when the automatic audit of database size occurs. Vertica performs this audit if your Vertica license includes a data size allowance. For details of this audit, see Managing Licenses in the Administrator's Guide. To change the time the audit runs, use the SET_AUDIT_TIME function.

Syntax

```
GET_AUDIT_TIME()
```

Privileges
None

Example

```sql
=> SELECT get_audit_time();
get_audit_time
--------------------
The audit is scheduled to run at 11:59 PM each day.
(1 row)
```
GET_COMPLIANCE_STATUS

Displays whether your database is in compliance with your Vertica license agreement. This information includes the results of Vertica's most recent audit of the database size (if your license has a data allowance as part of its terms), the license term (if your license has an end date), and the number of nodes (if your license has a node limit).

GET_COMPLIANCE_STATUS measures data allowance by TBs (where a TB equals $1024^4$ bytes).

The information displayed by GET_COMPLIANCE_STATUS includes:

- The estimated size of the database (see Auditing Database Size in the Administrator's Guide for an explanation of the size estimate).

- The raw data size allowed by your Vertica license.

- The percentage of your allowance that your database is currently using.

- The number of nodes and license limit.

- The date and time of the last audit.

- Whether your database complies with the data allowance terms of your license agreement.

- The end date of your license.

- How many days remain until your license expires.

Note: If your license does not have a data allowance, end date, or node limit, some of the values might not appear in the output for GET_COMPLIANCE_STATUS.

If the audit shows your license is not in compliance with your data allowance, you should either delete data to bring the size of the database under the licensed amount, or upgrade your license. If your license term has expired, you should contact Vertica immediately to renew your license. See Managing Licenses in the Administrator's Guide for further details.

Syntax

GET_COMPLIANCE_STATUS()

Privileges

None
Examples

```sql
=> SELECT GET_COMPLIANCE_STATUS();
get_compliance_status
-----------------------
Raw Data Size: 0.00TB +/- 0.00TB
License Size : 10.00TB
Utilization : 0%
Audit Time : 2015-09-24 12:19:15.425486-04
Compliance Status : The database is in compliance with respect to raw data size.

License End Date: 2015-11-23 00:00:00 Days Remaining: 60.53
(1 row)
```

The following example shows output for a Vertica for SQL on Apache Hadoop cluster.

```sql
=> SELECT GET_COMPLIANCE_STATUS();
get_compliance_status
-----------------------
Node count : 4
License Node limit : 5
No size-compliance concerns for an Unlimited license

No expiration date for a Perpetual license
(1 row)
```

SET_AUDIT_TIME

Sets the time that Vertica performs automatic database size audit to determine if the size of the database is compliant with the raw data allowance in your Vertica license. Use this function if the audits are currently scheduled to occur during your database's peak activity time. This is normally not a concern, since the automatic audit has little impact on database performance.

Audits are scheduled by the preceding audit, so changing the audit time does not affect the next scheduled audit. For example, if your next audit is scheduled to take place at 11:59PM and you use SET_AUDIT_TIME to change the audit schedule 3AM, the previously scheduled 11:59PM audit still runs. As that audit finishes, it schedules the next audit to occur at 3AM.

Vertica always performs the next scheduled audit even where you have changed the audit time using SET_AUDIT_TIME and then triggered an automatic audit by issuing the statement, SELECT AUDIT_LICENSE_SIZE. Only after the next scheduled audit does Vertica begin auditing at the new time you set using SET_AUDIT_TIME. Thereafter, Vertica audits at the new time.

Syntax

```
SET_AUDIT_TIME(time)
```
time | A string containing the time in 'HH:MM AM/PM' format (for example, '1:00 AM') when the audit should run daily.

Privileges
Superuser

Example

```sql
=> SELECT SET_AUDIT_TIME('3:00 AM');

<table>
<thead>
<tr>
<th>set_audit_time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------------</td>
<td>--------</td>
</tr>
</tbody>
</table>
```

The scheduled audit time will be set to 3:00 AM after the next audit.

(1 row)

Multiple Active Result Sets Functions

This section contains the functions associated with the Vertica library for Multiple Active Result Sets (MARS).

CLOSE_ALL_RESULTSETS

Closes all result set sessions within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.

Syntax

```sql
SELECT CLOSE_ALL_RESULTSETS ('session_id')
```

Parameters

| session_id | A string that specifies the Multiple Active Result Sets session. |

Privileges

None; however, without superuser privileges, you can only close your own session's results.
Examples

This example shows how you can view a MARS result set, then close the result set, and then confirm that the result set has been closed.

Query the MARS storage table. One session ID is open and three result sets appear in the output.

```sql
SELECT * FROM SESSION_MARS_STORE;
```

| node_name: v_vmart_node0001 | session_id: server1.company.-83046:1y28gu9 | remaining_row_count: 776460 | bytes_used: 89692848 | user_name: dbadmin | resultset_id: 7 | row_count: 777460 |
| node_name: v_vmart_node0001 | session_id: server1.company.-83046:1y28gu9 | remaining_row_count: 323349 | bytes_used: 81862010 | user_name: dbadmin | resultset_id: 8 | row_count: 324349 |
| node_name: v_vmart_node0001 | session_id: server1.company.-83046:1y28gu9 | remaining_row_count: 276947 | bytes_used: 32978280 | user_name: dbadmin | resultset_id: 9 | row_count: 277947 |

(1 row)

Close all result sets for session server1.company.-83046:1y28gu9:

```sql
SELECT CLOSE_ALL_RESULTSETS('server1.company.-83046:1y28gu9');
```

Closing all result sets from server1.company.-83046:1y28gu9

(1 row)

Query the MARS storage table again for the current status. You can see that the session and result sets have been closed:

```sql
SELECT * FROM SESSION_MARS_STORE;
```

| node_name: v_vmart_node0001 | session_id: server1.company.-83046:1y28gu9 | remaining_row_count: 0 | bytes_used: 0 | user_name: dbadmin | resultset_id: 0 | row_count: 0 |

(0 rows)

**CLOSE_RESULTSET**

Closes a specific result set within Multiple Active Result Sets (MARS) and frees the MARS storage for other result sets.
Syntax

```
SELECT CLOSE_RESULTSET ('session_id', ResultSetID)
```

Parameters

<table>
<thead>
<tr>
<th>session_id</th>
<th>A string that specifies the Multiple Active Result Sets session containing the ResultSetID to close.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResultSetID</td>
<td>An integer that specifies which result set to close.</td>
</tr>
</tbody>
</table>

Privileges

None; however, without superuser privileges, you can only close your own session's results.

Examples

This example shows a MARS storage table opened. One session_id is currently open, and one result set appears in the output.

```
SELECT * FROM SESSION_MARS_STORE;
```

```
<table>
<thead>
<tr>
<th>node_name</th>
<th>session_id</th>
<th>user_name</th>
<th>resultset_id</th>
<th>row_count</th>
<th>remaining_row_count</th>
<th>bytes_used</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>server1.company.-83046:1y28gu9</td>
<td>dbadmin</td>
<td>1</td>
<td>318718</td>
<td>312718</td>
<td>80441904</td>
</tr>
</tbody>
</table>
(1 row)
```

Close user session server1.company.-83046:1y28gu9 and result set 1:

```
SELECT CLOSE_RESULTSET('server1.company.-83046:1y28gu9', 1);
```

```
Closing result set 1 from server1.company.-83046:1y28gu9
(1 row)
```

Query the MARS storage table again for current status. You can see that result set 1 is now closed:

```
SELECT * FROM SESSION_MARS_STORE;
```

```
<table>
<thead>
<tr>
<th>node_name</th>
<th>session_id</th>
<th>user_name</th>
<th>resultset_id</th>
<th>row_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>server1.company.-83046:1y28gu9</td>
<td>dbadmin</td>
<td>1</td>
<td>318718</td>
</tr>
</tbody>
</table>
(1 row)
```
Partition Management Functions

This section contains partition management functions specific to Vertica.

CALENDAR_HIERARCHY_DAY

Specifies to group DATE partition keys into a hierarchy of years, months, and days. The Vertica Tuple Mover regularly evaluates partition keys against the current date, and merges partitions as needed into the appropriate year and month partition groups.

Syntax

CALENDAR_HIERARCHY_DAY( partition-expression[, active-months[, active-years] ] )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition-expression</td>
<td>The DATE expression on which to group partition keys, which must be identical to the table's PARTITION BY expression.</td>
</tr>
<tr>
<td>active-months</td>
<td>An integer ( \geq 0 ) that specifies how many months preceding ( \text{MONTH(CURRENT_DATE)} ) to store unique partition keys in separate partitions.</td>
</tr>
<tr>
<td></td>
<td>If you specify 1, only partition keys of the current month are stored in separate partitions.</td>
</tr>
<tr>
<td></td>
<td>If you specify 0, all partition keys of the current month are merged into a partition group for that month.</td>
</tr>
<tr>
<td></td>
<td>For details, see Hierarchical Partitioning.</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>active-years</td>
<td>An integer ( \geq 0 ), specifies how many years preceding ( \text{YEAR(CURRENT_DATE)} ) to partition group keys by month in separate partitions.</td>
</tr>
<tr>
<td></td>
<td>If you specify 1, only partition keys of the current year are stored in month partition groups.</td>
</tr>
<tr>
<td></td>
<td>If you specify 0, all partition keys of the current and previous years are merged into year partition groups.</td>
</tr>
</tbody>
</table>
For details, see Hierarchical Partitioning.
Default value: 2

Important: The CALENDAR_HIERARCHY_DAY algorithm assumes that most table activity is focused on recent dates. Setting active-years and active-months to a low number ≥ 2 serves to isolate merge activity to date-specific containers, and incurs minimal overhead. Vertica recommends that you use the default setting of 2 for active-years and active-months. For most users, these settings achieve an optimal balance between ROS storage and performance.

Usage

Specify this function in a table partition clause, as its GROUP BY expression:

```
PARTITION BY partition-expression
  GROUP BY CALENDAR_HIERARCHY_DAY(
    group-expression
    [, active-months[, active-years] ]
  )
```

For example:

```
=> CREATE TABLE public.store_orders
  (    order_no int,
       order_date timestamp NOT NULL,
       shipper varchar(20),
       ship_date date
  );
...
=> ALTER TABLE public.store_orders
  PARTITION BY order_date::DATE
  GROUP BY CALENDAR_HIERARCHY_DAY(order_date::DATE, 3, 2) REORGANIZE;
```

For details on usage, see Hierarchical Partitioning in the Administrator's Guide.

See Also

Hierarchical Partitioning in the Administrator's Guide

COPY_PARTITIONS_TO_TABLE

Copies partitions from one table to another. This lightweight partition copy increases performance by initially sharing the same storage between two tables. After the copy operation is complete, the tables are independent of each other. Users can perform operations
on one table without impacting the other. These operations can increase the overall storage required for both tables.

Note: Although they share storage space, Vertica considers the partitions as discrete objects for license capacity purposes. For example, copying a one TB partition would only consume one TB of space. Your Vertica license, however, considers them as separate objects consuming two TB of space.

Syntax

COPY_PARTITIONS_TO_TABLE (  
  '[[database.]schema.]source-table',  
  'min-range-value',  
  'max-range-value',  
  '[[database.]schema.]target-table'  
)  

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>source-table</td>
<td>The source table of the partitions to copy.</td>
</tr>
<tr>
<td>min-range-value</td>
<td>The minimum and maximum value of partition keys to copy, where min-range-value must be ≤ max-range-value. To copy one partition, min-range-value and max-range-value must be equal.</td>
</tr>
<tr>
<td>max-range-value</td>
<td></td>
</tr>
<tr>
<td>target-table</td>
<td>The target table of the partitions to copy. If the table does not exist, Vertica creates a table from the source table's definition, by calling CREATE TABLE with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table.</td>
</tr>
<tr>
<td>force-split</td>
<td>Optional Boolean argument, specifies whether to split ROS containers if the range of partition keys spans multiple containers or part of a single container:</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>true: Split ROS containers as needed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false (default): Return with an error if ROS containers must be split to implement this operation.</td>
</tr>
</tbody>
</table>

### Privileges

- Ownership or USAGE privileges on the source table.
- CREATE privileges on the target table, if COPY_PARTITIONS_TO_TABLE creates it.

### Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partition clause
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to Enforcing Primary Key and Foreign Key Constraints.

  Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

- Check constraints. For MOVE_PARTITIONS_TO_TABLE and COPY_PARTITIONS_TO_TABLE, Vertica enforces enabled check constraints on the target table only. For SWAP_PARTITIONS_BETWEEN_TABLES, Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.
- Number and definitions of text indices.

### Table Restrictions

The following restrictions apply to the source and target tables:
• If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.

• The following tables cannot be used as sources or targets:
  - Temporary tables
  - Virtual tables
  - System tables
  - External tables

Examples

If you call COPY_PARTITIONS_TO_TABLE and the target table does not exist, the function creates the table automatically. In the following example, the target table `partn_backup.trades_200801` does not exist. COPY_PARTITIONS_TO_TABLE creates the table and replicates the partition. Vertica also copies all the constraints associated with the source table except foreign key constraints.

```sql
=> SELECT COPY_PARTITIONS_TO_TABLE ('prod_trades', '200801', '200801', 'partn_backup.trades_200801');
COPY_PARTITIONS_TO_TABLE
----------------------------------------
1 distinct partition values copied at epoch 15.
(1 row)
```

See Also

Archiving Partitions

DROP_PARTITIONS

This function supersedes meta-function DROP_PARTITION. Vertica will continue to support DROP_PARTITION. For information about this function, refer to the Vertica 8.1 documentation.

Drops the specified table partition keys. If the WOS contains table data, DROP_PARTITIONS forces a moveout before it executes the drop operation.
Syntax

DROP_PARTITIONS (  
  '[[database.]schema.]table-name',  
  'min-range-value',  
  'max-range-value'  
  [, 'force-split']  
)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>table-name</td>
<td>The target table. The table cannot be used as a dimension table in a pre-join projection and cannot have out-of-date (unrefreshed) projections.</td>
</tr>
<tr>
<td>min-range-value</td>
<td>The minimum and maximum value of partition keys to drop, where min-range-value must be ≤ max-range-value. To drop one partition key, min-range-value and max-range-value must be equal.</td>
</tr>
<tr>
<td>max-range-value</td>
<td></td>
</tr>
</tbody>
</table>
| force-split     | Optional Boolean argument, specifies whether to split ROS containers if the range of partition keys spans multiple containers or part of a single container:  
  • true: Split ROS containers as needed.  
  • false (default): Return with an error if ROS containers must be split to implement this operation. |

Privileges

One of the following:
- DBADMIN
- Table owner
- USAGE privileges on the table schema and TRUNCATE privileges on the table

Examples

See [Dropping Partitions](#) in the Administrator's Guide.

See Also

*PARTITION_TABLE*

## DUMP_PROJECTION_PARTITION_KEYS

Dumps the partition keys of the specified projection.

### Syntax

```
DUMP_PROJECTION_PARTITION_KEYS( '{[database.]schema.]projection-name'}
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td>projection-name</td>
<td>The projection name.</td>
</tr>
</tbody>
</table>

### Privileges

- SELECT privilege on table
- USAGE privileges on schema
Example

The following statements create the table and projection `online_sales.online_sales_fact` and `online_sales.online_sales_fact_unseg_p`, respectively, and partitions table data by the column `call_center_key`:

```sql
=> CREATE TABLE online_sales.online_sales_fact
(
    sale_date_key int NOT NULL,
    ship_date_key int NOT NULL,
    product_key int NOT NULL,
    product_version int NOT NULL,
    customer_key int NOT NULL,
    call_center_key int NOT NULL,
    online_page_key int NOT NULL,
    shipping_key int NOT NULL,
    warehouse_key int NOT NULL,
    promotion_key int NOT NULL,
    pos_transaction_number int NOT NULL,
    sales_quantity int,
    sales_dollar_amount float,
    ship_dollar_amount float,
    net_dollar_amount float,
    cost_dollar_amount float,
    gross_profit_dollar_amount float,
    transaction_type varchar(16)
) PARTITION BY (online_sales_fact.call_center_key);
```

The following `DUMP_PROJECTION_PARTITION_KEYS` statement dumps the partition key from the projection `online_sales.online_sales_fact_unseg_p`:

```sql
=> SELECT DUMP_PROJECTION_PARTITION_KEYS('online_sales.online_sales_fact_unseg_p');
```

Partition keys on node `v_vmart_node0001`
Projection 'online_sales_fact_unseg_p'
Storage [ROS container]
   No of partition keys: 1
   Partition keys: 200
Storage [ROS container]
   No of partition keys: 1
   Partition keys: 199
...  
Storage [ROS container]
   No of partition keys: 1
   Partition keys: 2
Storage [ROS container]
   No of partition keys: 1
   Partition keys: 1

Partition keys on node `v_vmart_node0002`
Projection 'online_sales_fact_unseg_p'

...
See Also

- Partitioning Tables in the Administrator's Guide
- DUMP_PARTITION_KEYS
- DUMP_TABLE_PARTITION_KEYS
- PARTITION_PROJECTION
- PARTITION_TABLE

**DUMP_TABLE_PARTITION_KEYS**

Dumps the partition keys of all projections for the specified table.

**Syntax**

```
DUMP_TABLE_PARTITION_KEYS ( 'table-name' )
```

**Parameters**

<table>
<thead>
<tr>
<th>table-name</th>
<th>The name of the table.</th>
</tr>
</thead>
</table>

**Privilege**

- SELECT privilege on table
- USAGE privileges on schema

**Examples**

The following example creates a simple table called states and partitions the data by state:
Now dump the partition keys of all projections anchored on table `states`:

```sql
=> SELECT DUMP_TABLE_PARTITION_KEYS('states');

Partition keys on node v_vmart_node0001
Projection 'states_p'
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: VT
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: PA
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: NY
  Storage [ROS container]
    No of partition keys: 1
    Partition keys: MA

Partition keys on node v_vmart_node0002
...
(1 row)
```

See Also

- `DUMP_PROJECTION_PARTITION_KEYS`
- `DUMP_TABLE_PARTITION_KEYS`
- `PARTITION_PROJECTION`
- `PARTITION_TABLE`
- `Partitioning Tables` in the Administrator's Guide

**MOVE_PARTITIONS_TO_TABLE**

Moves partitions from one table to another.
Syntax

MOVE_PARTITIONS_TO_TABLE ( 
  '[[database.]schema.]source-table',
  'min-range-value',
  'max-range-value',
  '[[database.]schema.]target-table' 
  [, force-split] 
)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>source-table</code></td>
<td>The source table of the partitions to move.</td>
</tr>
<tr>
<td><code>min-range-value</code></td>
<td>The minimum and maximum value of partition keys to move, where <code>min-range-value</code> must be (\leq max-range-value). To move one partition, <code>min-range-value</code> and <code>max-range-value</code> must be equal.</td>
</tr>
<tr>
<td><code>max-range-value</code></td>
<td></td>
</tr>
<tr>
<td><code>target-table</code></td>
<td>The target table of the partitions to move. If the table does not exist, Vertica creates a table from the source table's definition, by calling <code>CREATE TABLE</code> with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table.</td>
</tr>
<tr>
<td><code>force-split</code></td>
<td>Optional Boolean argument, specifies whether to split ROS containers if the range of partition keys spans multiple containers or part of a single container:</td>
</tr>
<tr>
<td></td>
<td>* true: Split ROS containers as needed.</td>
</tr>
<tr>
<td></td>
<td>* false (default): Return with an error if ROS containers must be split to implement this operation.</td>
</tr>
</tbody>
</table>
Privileges

If the target table does not exist, you must have CREATE privileges on the target schema, to enable table creation. One of the following conditions is also required:

- DBADMIN role
- Owner of the source and target tables
- USAGE privileges on source and target schemas, TRUNCATE privileges on the source table, and INSERT privileges on the target table

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partition clause
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to Enforcing Primary Key and Foreign Key Constraints.

  Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

- Check constraints. For MOVE_PARTITIONS_TO_TABLE and COPY_PARTITIONS_TO_TABLE, Vertica enforces enabled check constraints on the target table only. For SWAP_PARTITIONS_BETWEEN_TABLES, Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.

- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:
If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.

The following tables cannot be used as sources or targets:
- Temporary tables
- Virtual tables
- System tables
- External tables

Examples
See Archiving Partitions.

See Also
- COPY_PARTITIONS_TO_TABLE
- SWAP_PARTITIONS_BETWEEN_TABLES

PARTITION_PROJECTION

Splits ROS containers for a specified projection. PARTITION_PROJECTION also purges data while partitioning ROS containers if deletes were applied before the AHM epoch.

Syntax

```
PARTITION_PROJECTION ( '[[database.]schema.]projection')
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>
projection | The projection to partition.

Privileges

- Table owner
- USAGE privilege on schema

Examples

In this example, PARTITION_PROJECTION forces a split of ROS containers on the `states_p` projection:

```sql
=> SELECT PARTITION_PROJECTION ('states_p');
PARTITION_PROJECTION
------------------------
Projection partitioned
(1 row)
```

See Also

- PARTITION_TABLE
- Partitioning Tables in the Administrator's Guide

PARTITION_TABLE

Invokes the Tuple Mover to reorganize ROS storage containers as needed to conform with the current partitioning policy.

Syntax

```
PARTITION_TABLE ( '[schema.]table-name')
```

Parameters

<table>
<thead>
<tr>
<th><code>[database.]schema</code></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
</table>
If you specify a database, it must be the current database.

| table-name | The table to partition. |

Privileges

- Table owner
- USAGE privilege on schema

Restrictions

You cannot run PARTITION_TABLE on a table that is an anchor table for a live aggregate projection or a Top-K projection.

See Also

- PARTITION_PROJECTION
- Partitioning Existing Table Data

**PURGE_PARTITION**

Purges a table partition of deleted rows. Similar to PURGE and PURGE_PROJECTION, this function removes deleted data from physical storage so you can reuse the disk space. PURGE_PARTITION removes data only from the AHM epoch and earlier.

Syntax

```sql
PURGE_PARTITION ( '[database.]schema.]table', partition-key )
```

Parameters

| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: |

If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>table</th>
<th>The partitioned table to purge.</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition-key</td>
<td>The key of the partition to purge.</td>
</tr>
</tbody>
</table>

**Privileges**

- Table owner
- USAGE privilege on schema

**Example**

The following example lists the count of deleted rows for each partition in a table, then calls PURGE_PARTITION() to purge the deleted rows from the data.

```sql
=> SELECT partition_key, table_schema, projection_name, sum(deleted_row_count)
    AS deleted_row_count FROM partitions
GROUP BY partition_key, table_schema, projection_name
ORDER BY partition_key;
```

<table>
<thead>
<tr>
<th>partition_key</th>
<th>table_schema</th>
<th>projection_name</th>
<th>deleted_row_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>public</td>
<td>t_super</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>public</td>
<td>t_super</td>
<td>1</td>
</tr>
</tbody>
</table>

(10 rows)

```sql
=> SELECT PURGE_PARTITION('t',5); -- Purge partition with key 5.
```

**Task:** merge partitions

(Table: public.t) (Projection: public.t_super)
(1 row)

```sql
=> SELECT partition_key, table_schema, projection_name, sum(deleted_row_count)
    AS deleted_row_count FROM partitions
GROUP BY partition_key, table_schema, projection_name
ORDER BY partition_key;
```

<table>
<thead>
<tr>
<th>partition_key</th>
<th>table_schema</th>
<th>projection_name</th>
<th>deleted_row_count</th>
</tr>
</thead>
</table>
See Also

- `PURGE`  
- `PURGE_PROJECTION`  
- `PURGE_TABLE`  
- `STORAGE_CONTAINERS`

**SWAP_PARTITIONS_BETWEEN_TABLES**

Swaps partitions between two tables.

**Syntax**

```
SWAP_PARTITIONS_BETWEEN_TABLES (  
    '[[database.]schema.]staging-table',  
    'min-range-value',  
    'max-range-value',  
    '[[database.]schema.]target-table'  
    [, force-split]  
)
```

**Parameters**

<table>
<thead>
<tr>
<th><code>[[database.]schema]</code></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>myschema.thisDbObject</td>
<td>(Example)</td>
</tr>
</tbody>
</table>
If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>staging-table</td>
<td>The staging table from which to swap partitions.</td>
</tr>
<tr>
<td>min-range-value</td>
<td>The minimum and maximum value of partition keys to swap,</td>
</tr>
<tr>
<td>max-range-value</td>
<td>where min-range-value must be ( \leq ) max-range-value. To swap one partition, min-range-value and max-range-value must be equal.</td>
</tr>
<tr>
<td>target-table</td>
<td>The table to which the partitions are to be swapped. The target table cannot be the same as the staging table.</td>
</tr>
<tr>
<td>force-split</td>
<td>Optional Boolean argument, specifies whether to split ROS containers if the range of partition keys spans multiple containers or part of a single container:</td>
</tr>
<tr>
<td></td>
<td>- true: Split ROS containers as needed.</td>
</tr>
<tr>
<td></td>
<td>- false (default): Return with an error if ROS containers must be split to implement this operation.</td>
</tr>
</tbody>
</table>

**Privileges**

One of the following:

- DBADMIN role
- Owner of both tables
- USAGE privileges on both schemas, and TRUNCATE and INSERT privileges on both tables.

**Requirements**

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partition clause
- Number of projections
- Projection sort order
• Primary and unique key constraints. However, the key constraints do not have to be identically enabled. For more information on constraints, refer to Enforcing Primary Key and Foreign Key Constraints.

  Note: If the target table has primary or unique key constraints enabled and copying or moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation.

• Check constraints. For MOVE_PARTITIONS_TO_TABLE and COPY_PARTITIONS_TO_TABLE, Vertica enforces enabled check constraints on the target table only. For SWAP_PARTITIONS_BETWEEN_TABLES, Vertica enforces enabled check constraints on both tables. If there is a violation of an enabled check constraint, Vertica rolls back the operation.

• Number and definitions of text indices.

Restrictions

The following restrictions apply to the source and target tables:

• If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.

• The following tables cannot be used as sources or targets:
  - Temporary tables
  - Virtual tables
  - System tables
  - External tables

Examples

See Swapping Partitions.

See Also

MOVE_PARTITIONS_TO_TABLE
Profiling Functions

This section contains profiling functions specific to Vertica.

CLEAR_PROFILING

Clears from memory data for the specified profiling type.

*Note:* Vertica stores profiled data in memory, so profiling can be memory intensive depending on how much data you collect.

Syntax

`CLEAR_PROFILING( 'profiling-type' )`

Parameters

<table>
<thead>
<tr>
<th>profiling-type</th>
<th>The type of profiling data to clear:</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Clears profiling for basic session parameters and lock time out data.</td>
</tr>
<tr>
<td>query</td>
<td>Clears profiling for general information about queries that ran, such as the query strings used and the duration of queries.</td>
</tr>
<tr>
<td>ee</td>
<td>Clears profiling for information about the execution run of each query.</td>
</tr>
</tbody>
</table>

Example

The following statement clears profiled data for queries:

```
=> SELECT CLEAR_PROFILING('query');
```
See Also

- DISABLE_PROFILING
- ENABLE_PROFILING
- SHOW_PROFILING_CONFIG
- Profiling Database Performance

DISABLE_PROFILING

Disables profiling for the specified profiling type.

Syntax

DISABLE_PROFILING( 'profiling-type' )

Parameters

<table>
<thead>
<tr>
<th>profiling-type</th>
<th>The type of profiling data to disable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Disables profiling for basic session parameters and lock time out data.</td>
</tr>
<tr>
<td>query</td>
<td>Disables profiling for general information about queries that ran, such as the query strings used and the duration of queries.</td>
</tr>
<tr>
<td>ee</td>
<td>Disables profiling for information about the execution run of each query.</td>
</tr>
</tbody>
</table>

Example

The following statement disables profiling on query execution runs:

```sql
=> SELECT DISABLE_PROFILING('ee');
DISABLE_PROFILING
---------------------
EE Profiling Disabled
(1 row)
```
See Also

- CLEAR_PROFILING
- ENABLE_PROFILING
- SHOW_PROFILING_CONFIG
- Profiling Database Performance

ENABLE_PROFILING

Enables profiling for the specified profiling type.

**Note:** Vertica stores profiled data in memory, so profiling can be memory intensive depending on how much data you collect.

Syntax

```
ENABLE_PROFILING( 'profiling-type' )
```

Parameters

<table>
<thead>
<tr>
<th>profiling-type</th>
<th>The type of profiling data to enable:</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Enables profiling for basic session parameters and lock timeout data.</td>
</tr>
<tr>
<td>query</td>
<td>Enables profiling for general information about queries that ran, such as the query strings used and the duration of queries.</td>
</tr>
<tr>
<td>ee</td>
<td>Enables profiling for information about the execution run of each query.</td>
</tr>
</tbody>
</table>

Example

The following statement enables profiling on query execution runs:

```
=> SELECT ENABLE_PROFILING('ee');
```

Enable PROFILING
SHOW_PROFILING_CONFIG

Shows whether profiling is enabled.

Syntax

SHOW_PROFILING_CONFIG ()

Example

The following statement shows that profiling is enabled globally for all profiling types (session, execution engine, and query):

```sql
=> SELECT SHOW_PROFILING_CONFIG();
SHOW_PROFILING_CONFIG
-----------------------------
Session Profiling: Session off, Global on
EE Profiling: Session off, Global on
Query Profiling: Session off, Global on
(1 row)
```

See Also

- CLEAR_PROFILING
- DISABLE_PROFILING
- SHOW_PROFILING_CONFIG
- Profiling Database Performance
• **ENABLE_PROFILING**

• **Profiling Database Performance**
Projection Management Functions

This section contains projection management functions specific to Vertica.

See Also

- V_CATALOG.PROJECTIONS
- V_CATALOG.PROJECTION_COLUMNS
- V_MONITOR.PROJECTION_REFRESHES
- V_MONITOR.PROJECTION_STORAGE

CLEAR_PROJECTION_REFRESHES

Clears information in system table PROJECTION_REFRESHES of projection refresh history.

System table PROJECTION_REFRESHES records information about refresh operations, successful and unsuccessful. PROJECTION_REFRESHES retains refresh data until one of the following events occurs:

- CLEAR_PROJECTION_REFRESHES is called.
- The table's storage quota is exceeded.

PROJECTION_REFRESHES checks the Boolean column IS_EXECUTING in PROJECTION_REFRESHES to determine whether refresh operations are still running or are complete. The function only removes information for refresh operations that are complete.

Syntax

CLEAR_PROJECTION_REFRESHES()

Privileges

Superuser
Example

```sql
=> SELECT CLEAR_PROJECTION_REFRESHES();
CLEAR_PROJECTION_REFRESHES
-----------------------------
CLEAR
(1 row)
```

See Also

- `REFRESH`
- `START_REFRESH`
- Clearing Projection Refresh History

**EVALUATE_DELETE_PERFORMANCE**

Evaluates projections for potential `DELETE` performance issues. If there are issues found, a warning message is displayed. For steps you can take to resolve delete and update performance issues, see `DELETE` and `UPDATE Optimization` in the Administrator's Guide. This function uses data sampling to determine whether there are any issues with a projection. Therefore, it does not generate false-positives warnings, but it can miss some cases where there are performance issues.

**Note:** Optimizing for delete performance is the same as optimizing for update performance. So, you can use this function to help optimize a projection for updates as well as deletes.

Syntax

`EVALUATE_DELETE_PERFORMANCE ( 'target' )`

Parameters

| `target` | The name of a projection or table. If you supply the name of a projection, only that projection is evaluated for `DELETE` performance issues. If you supply the name of a table, then all of the projections anchored to the table will be evaluated for issues. If you do not provide a projection or table name, `EVALUATE_DELETE_` |
PERFORMANCE examines all of the projections that you can access for DELETE performance issues. Depending on the size you your database, this may take a long time.

Privileges
None

Notes
When evaluating multiple projections, EVALUATE_DELETE_PERFORMANCE reports up to ten projections that have issues, and refers you to a table that contains the full list of issues it has found.

Example
The following example demonstrates how you can use EVALUATE_DELETE_PERFORMANCE to evaluate your projections for slow DELETE performance.

```sql
=> create table example (A int, B int,C int);
CREATE TABLE
=> create projection one_sort (A,B,C) as (select A,B,C from example) order by A;
CREATE PROJECTION
=> create projection two_sort (A,B,C) as (select A,B,C from example) order by A,B;
CREATE PROJECTION
=> select evaluate_delete_performance('one_sort');
   evaluate_delete_performance
No projection delete performance concerns found.
(1 row)
=> select evaluate_delete_performance('two_sort');
   evaluate_delete_performance
No projection delete performance concerns found.
(1 row)
```

The previous example showed that there was no structural issue with the projection that would cause poor DELETE performance. However, the data contained within the projection can create potential delete issues if the sorted columns do not uniquely identify a row or small number of rows. In the following example, Perl is used to populate the table with data using a nested series of loops. The inner loop populates column C, the middle loop populates column B, and the outer loop populates column A. The result is column A contains only three distinct values (0, 1, and 2), while column B slowly varies between 20 and 0 and column C changes in
each row. EVALUATE_DELETE_PERFORMANCE is run against the projections again to see if the data within the projections causes any potential DELETE performance issues.

```plaintext
=> \! perl -e 'for ($i=0; $i<3; $i++) { for ($j=0; $j<21; $j++) { for ($k=0; $k<19; $k++) { printf "%d,%d,%d\n", $i,$j,$k;}}}' | /opt/vertica/bin/vsql -c "copy example from stdin delimiter ',' direct;"
Password:
=> select * from example;
A | B | C
---+---+---
0 | 20 | 18
0 | 20 | 17
0 | 20 | 16
0 | 20 | 15
0 | 20 | 14
0 | 20 | 13
0 | 20 | 12
0 | 20 | 11
0 | 20 | 10
0 | 20 | 9
0 | 20 | 8
0 | 20 | 7
0 | 20 | 6
0 | 20 | 5
0 | 20 | 4
0 | 20 | 3
0 | 20 | 2
0 | 20 | 1
0 | 20 | 0
0 | 19 | 18
1157 rows omitted
2 | 1 | 0
2 | 0 | 18
2 | 0 | 17
2 | 0 | 16
2 | 0 | 15
2 | 0 | 14
2 | 0 | 13
2 | 0 | 12
2 | 0 | 11
2 | 0 | 10
2 | 0 | 9
2 | 0 | 8
2 | 0 | 7
2 | 0 | 6
2 | 0 | 5
2 | 0 | 4
2 | 0 | 3
2 | 0 | 2
2 | 0 | 1
2 | 0 | 0
=> SELECT COUNT (*) FROM example;
COUNT
-------
1197
(1 row)
=> SELECT COUNT (DISTINCT A) FROM example;
COUNT
-------
```
The one_sort projection has potential delete issues since it only sorts on column A which has few distinct values. This means that each value in the sort column corresponds to many rows in the projection, which negatively impacts DELETE performance. Since the two_sort projection is sorted on columns A and B, each combination of values in the two sort columns identifies just a few rows, allowing deletes to be performed faster.

Not supplying a projection name results in all of the projections you can access being evaluated for DELETE performance issues.

GET_PROJECTION_STATUS

Returns information relevant to the status of a projection:

- The current K-safety status of the database
- The number of nodes in the database
- Whether the projection is segmented
- The number and names of buddy projections
- Whether the projection is safe
- Whether the projection is up to date
- Whether statistics have been computed for the projection

Use GET_PROJECTION_STATUS to monitor the progress of a projection data refresh.
Syntax

GET_PROJECTION_STATUS ( '[[database.]schema.]projection' );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| [database.]schema  | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: |\
|                    | mysql.schema.thisDbObject                                                   |
| projection         | The projection for which to display status.                                 |

Examples

=> SELECT GET_PROJECTION_STATUS('public.customer_dimension_site01');

---

Current system K is 1.
# of Nodes: 4.
public.customer_dimension_site01 [Segmented: No] [Seg Cols: ] [K: 3] [public.customer_dimension_site04, public.customer_dimension_site03, public.customer_dimension_site02] [Safe: Yes] [UptoDate: Yes] [Stats: Yes]

GET_PROJECTIONS

Returns the following information about projections of the specified anchor table:

<table>
<thead>
<tr>
<th>Contextual information</th>
<th>Projection data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database K-safety</td>
<td>For each projection, specifies:</td>
</tr>
<tr>
<td>Number of database nodes</td>
<td>All buddy projections</td>
</tr>
<tr>
<td>Number of projections for this table</td>
<td>Whether it is segmented</td>
</tr>
<tr>
<td></td>
<td>Whether it is safe</td>
</tr>
<tr>
<td></td>
<td>Whether it is up-to-date.</td>
</tr>
</tbody>
</table>
You can use GET_PROJECTIONS to monitor the progress of a projection data refresh.

Syntax

```
GET_PROJECTIONS ( '[[database.]schema-name.]table' )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `[database.]schema` | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:  

```
myschema.thisDbObject
```

If you specify a database, it must be the current database. |
| `table`         | Anchor table of the projections to list.                                    |

Privileges

None

Examples

The following example gets information about projections for VMart table `store.store_dimension`:

```
=> SELECT GET_PROJECTIONS('store.store_dimension');
-[ RECORD 1 ]-----
GET_PROJECTIONS | Current system K is 1.
# of Nodes: 3. 
Table store.store_dimension has 2 projections.

Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]
-----------------------------------------------------------------------------------------------
store.store_dimension_b1 [Segmented: Yes] [Seg Cols: "store.store_dimension.store_key"] [K: 1]  
[store.store_dimension_b0] [Safe: Yes] [UptoDate: Yes] [Stats: RowCounts]
store.store_dimension_b0 [Segmented: Yes] [Seg Cols: "store.store_dimension.store_key"] [K: 1]  
[store.store_dimension_b1] [Safe: Yes] [UptoDate: Yes] [Stats: RowCounts]
```

REFRESH

Synchronously refreshes one or more table projections in the foreground, and updates system table `PROJECTION_REFRESHES`. If you run REFRESH with no arguments, it refreshes all
projections that contain stale data.

Syntax

```
REFRESH ( [ '[[database.]schema.]table-name[,...]' ] )
```

Parameters

| `database.]schema` | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:
| `myschema.thisDbObject` |

If you specify a database, it must be the current database.

| `table-name` | The anchor table of the projections to refresh. If you specify multiple tables, REFRESH attempts to refresh them in parallel. Such calls are part of the Database Designer deployment (and deployment script). |

Returns

Note: If REFRESH does not refresh any projections, it returns a header string with no results.

<table>
<thead>
<tr>
<th>Column...</th>
<th>Returns...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection Name</td>
<td>The projection targeted for refresh.</td>
</tr>
<tr>
<td>Anchor Table</td>
<td>The projection's associated anchor table.</td>
</tr>
<tr>
<td>Status</td>
<td>Projections' refresh status:</td>
</tr>
<tr>
<td></td>
<td>• queued: Queued for refresh.</td>
</tr>
<tr>
<td></td>
<td>• refreshing: Refresh is in process.</td>
</tr>
<tr>
<td></td>
<td>• refreshed: Refresh successfully completed.</td>
</tr>
<tr>
<td></td>
<td>• failed: Refresh did not successfully complete.</td>
</tr>
<tr>
<td>Refresh Method</td>
<td>Method used to refresh the projection.</td>
</tr>
</tbody>
</table>
Privileges

- Superuser
- Owner of the specified tables

Refresh Methods

Vertica can refresh a projection from one of its buddies, if one is available. In this case, the target projection gets the source buddy's historical data. Otherwise, the projection is refreshed from scratch with data of the latest epoch at the time of the refresh operation. In this case, the projection cannot participate in historical queries on any epoch that precedes the refresh operation.

To determine the method used to refresh a given projection, query REFRESH_METHOD from system table PROJECTION_REFRESHES.

Examples

The following example refreshes the projections in tables t1 and t2:

```sql
=> SELECT REFRESH('t1, t2');

REFRESH

Refresh completed with the following outcomes:

Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]

"public"."t1_p": [t1] [refreshed] [scratch] [0] [0]"public"."t2_p": [t2] [refreshed] [scratch] [0] [0]
```

This next example shows that only the projection on table t was refreshed:

```sql
=> SELECT REFRESH('allow, public.deny, t');

REFRESH

Refresh completed with the following outcomes:

Projection Name: [Anchor Table] [Status] [Refresh Method] [Error Count] [Duration (sec)]

"n/a"."n/a": [n/a] [failed: insufficient permissions on table "allow"] [0] [1] [0]
```
REFRESH_COLUMNS

Refreshes table columns that are defined with the constraint SET USING. All refresh operations associated with a REFRESH_COLUMNS operation belong to the same transaction. All tables and columns specified by REFRESH_COLUMNS must be refreshed; otherwise, the entire operation is rolled back.

Syntax

```
REFRESH_COLUMNS ( 'tables', ['columns'] [ , 'refresh-mode' ] )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tables</strong></td>
<td>A comma-delimited list that specifies the tables to refresh:</td>
</tr>
<tr>
<td></td>
<td><code>[[database.]schema.]table[, ...]</code></td>
</tr>
<tr>
<td></td>
<td><strong>Important:</strong> If you specify multiple tables, you must also set parameter <code>refresh-mode</code> to REBUILD.</td>
</tr>
<tr>
<td><strong>columns</strong></td>
<td>Specifies one or more columns to refresh, as follows:</td>
</tr>
<tr>
<td></td>
<td>- '' (empty string)</td>
</tr>
<tr>
<td></td>
<td>Refresh all SET USING/DEFAULT USING columns in the specified tables.</td>
</tr>
<tr>
<td></td>
<td>- <code>[[database.]schema.]table.]column[, ...]</code></td>
</tr>
<tr>
<td></td>
<td>Refresh all columns in the comma-delimited list. The following requirements apply:</td>
</tr>
</tbody>
</table>
|             |     - If REFRESH_COLUMNS specifies multiple tables, all column names must be qualified by their table names. If the target tables span
multiple schemas, all column names must be fully qualified by their schema and table names.

- All specified columns must have a SET USING or DEFAULT USING constraint.

For example:

```
SELECT REFRESH_COLUMNS ('t1, t2', 't1.a, t2.b', 'REBUILD');
```

- `[[database.]schema.]table.*`
  Refresh all SET USING/DEFAULT USING columns in `table`. For example:

```
SELECT REFRESH_COLUMNS ('t1, t2', 't1.*, t2.b', 'REBUILD');
```

If you specify a database, it must be the current database.

**refresh-mode**

Specifies how to refresh SET USING columns:

- **UPDATE**: Marks original rows as deleted and replaces them with new rows. In order to save these updates, you must issue a COMMIT statement.

- **REBUILD**: Replaces all data in the specified columns. The rebuild operation is auto-committed.

If set to an empty string or omitted, REFRESH_COLUMNS executes in UPDATE mode. If you specify multiple tables, you must explicitly specify REBUILD mode.

In both cases, REFRESH_COLUMNS returns an error if any SET USING column is defined as a primary or unique key in a table that enforces those constraints.

See **REBUILD Mode Restrictions** for limitations on using the REBUILD option.

**Privileges**

- MODIFY privilege on the target table, USAGE privilege on its schema

- For each SET USING column to refresh that queries another table or view: SELECT privilege on the queried table/view, USAGE privilege on its schema
REBUILD versus REFRESH Modes

In general, UPDATE is a better choice when changes to SET USING column data are confined to a relatively small number of rows. Use REBUILD when a significant amount of SET USING column data is stale and must be updated. It is generally good practice to call REFRESH_COLUMNS with REBUILD on any new SET USING column—for example, to populate a SET USING column after adding it with ALTER TABLE...ADD COLUMN.

REBUILD Mode Restrictions

If you call REFRESH_COLUMNS on a SET USING column and specify the refresh mode as REBUILD, Vertica returns an error if any of the following conditions is true for that column:

- Specified as a table partition key.
- Included in a live aggregate projection or projection with expressions.
- Included in a projection's sort order or segmentation.
- Included in a projection, and the projection omits an anchor table column that is referenced in the column's SET USING expression.
- Included in a projection's GROUPED clause.

See Also

- Column-Constraint
- Defining Column Values

START_REFRESH

Refreshes projections in the current schema with the latest data of their respective anchor tables. START_REFRESH runs asynchronously in the background, and updates system table PROJECTION_REFRESHES. This function has no effect if a refresh is already running.

To refresh only projections of a specific table, use REFRESH. When you deploy a design through Database Designer, it automatically refreshes its projections.

Syntax

START_REFRESH()
Privileges
None

Requirements
All nodes must be up.

Refresh Methods
Vertica can refresh a projection from one of its buddies, if one is available. In this case, the target projection gets the source buddy's historical data. Otherwise, the projection is refreshed from scratch with data of the latest epoch at the time of the refresh operation. In this case, the projection cannot participate in historical queries on any epoch that precedes the refresh operation.

To determine the method used to refresh a given projection, query REFRESH_METHOD from system table PROJECTION_REFRESHES.

Example

```sql
=> SELECT START_REFRESH();
START_REFRESH
---------------------
Starting refresh background process.
(1 row)
```

See Also

- Refreshing Projections
- CLEAR_PROJECTION_REFRESHES
Purge Functions

This section contains purge functions specific to Vertica.

PURGE

Permanently removes delete vectors from ROS storage containers so disk space can be reused. PURGE removes all historical data up to and including the Ancient History Mark epoch.

PURGE does not delete temporary tables.

Caution: PURGE can temporarily take up significant disk space.

Syntax

PURGE()

Privileges

- Table owner
- USAGE privilege on schema

See Also

- PURGE_PROJECTION
- PURGE_TABLE
- Purging Deleted Data

PURGE_PROJECTION

Permanently removes deleted data from physical storage so disk space can be reused. You can purge historical data up to and including the Ancient History Mark epoch.

Caution: PURGE_PROJECTION can use significant disk space while purging the data.

See PURGE for details about purge operations.
Syntax

PURGE_PROJECTION ( '[[database.]schema.]projection' )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>projection</td>
<td>The projection to purge.</td>
</tr>
</tbody>
</table>

Privileges

- Table owner
- USAGE privilege on schema

Examples

The following example purges all historical data in projection tbl_p that precedes the Ancient History Mark epoch.

```sql
=> CREATE TABLE tbl (x int, y int);
CREATE TABLE
=> INSERT INTO tbl VALUES(1,2);
  OUTPUT
  -------
  1
  (1 row)
=> INSERT INTO tbl VALUES(3,4);
  OUTPUT
  -------
  1
  (1 row)
dbadmin=> COMMIT;
COMMl
=> CREATE PROJECTION tbl_p AS SELECT x FROM tbl UNSEGMENTED ALL NODES;
WARNING 4468: Projection <public.tbl_p> is not available for query processing.
```
Execute the select start_refresh() function to copy data into this projection. The projection must have a sufficient number of buddy projections and all nodes must be up before starting a refresh.

```sql
CREATE PROJECTION
=> SELECT START_REFRESH();
START_REFRESH

Starting refresh background process.
=> DELETE FROM tbl WHERE x=1;
OUTPUT
-------
 1
(1 row)

=> COMMIT;
COMMIT
=> SELECT MAKE_AHM_NOW();
MAKE_AHM_NOW

AHM set (New AHM Epoch: 9066)
(1 row)

=> SELECT PURGE_PROJECTION ('tbl_p');
PURGE_PROJECTION

Projection purged
(1 row)
```

See Also

- **PURGE_TABLE**
- **STORAGE_CONTAINERS**
- Purging Deleted Data in the Administrator's Guide.

**PURGE_TABLE**

Note: This function was formerly named PURGE_TABLE_PROJECTIONS(). Vertica still supports the former function name.

Permanently removes deleted data from physical storage so disk space can be reused. You can purge historical data up to and including the Ancient History Mark epoch.

Purges all projections of the specified table. You cannot use this function to purge temporary tables.

**Syntax**

```sql
PURGE_TABLE ( '[database.]schema.]table' )
```
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>&lt;myschema.thisDbObject&gt;</code></td>
</tr>
<tr>
<td><code>table</code></td>
<td>The table to purge.</td>
</tr>
</tbody>
</table>

Privileges

- Table owner
- USAGE privilege on schema

Caution: PURGE_TABLE could temporarily take up significant disk space while the data is being purged.

Example

The following example purges all projections for the store sales fact table located in the Vmart schema:

```sql
=> SELECT PURGE_TABLE('store.store_sales_fact');
```

See Also

- PURGE
- PURGE_TABLE
- STORAGE_CONTAINERS
- Purging Deleted Data
Session Management Functions

This section contains session management functions specific to Vertica.
See also the SQL system table V_MONITOR.SESSIONS.

CANCEL_REFRESH

Cancels refresh-related internal operations initiated by START_REFRESH and REFRESH.

Syntax

CANCEL_REFRESH()

Privileges

None

Notes

- Refresh tasks run in a background thread in an internal session, so you cannot use INTERRUPT_STATEMENT to cancel those statements. Instead, use CANCEL_REFRESH to cancel statements that are run by refresh-related internal sessions.

- Run CANCEL_REFRESH() on the same node on which START_REFRESH() was initiated.

- CANCEL_REFRESH() cancels the refresh operation running on a node, waits for the cancelation to complete, and returns SUCCESS.

- Only one set of refresh operations runs on a node at any time.

Example

Cancel a refresh operation executing in the background.

```sql
=> SELECT START_REFRESH();
START_REFRESH
-----------------------------------------------
Starting refresh background process.
```
(1 row)
=> SELECT CANCEL_REFRESH();
  CANCEL_REFRESH
-----------------------------------------------
Stopping background refresh process.
(1 row)

See Also

- INTERRUPT_STATEMENT
- SESSIONS
- START_REFRESH
- PROJECTION_REFRESHES

CLOSE_ALL_SESSIONS

Closes all external sessions except the one that issues this function. Use CLOSE_ALL_SESSIONS before shutting down the Vertica database.

Vertica closes sessions asynchronously, so it is possible for another session to open before all previous sessions close. In this case, you must reissue this function before the database can be shut down. To view the status of all open sessions, query the system table SESSIONS.

Query to view all database sessions.

For detailed information about session management options, see Managing Sessions in the Administrator's Guide.

Syntax

CLOSE_ALL_SESSIONS()

Privileges

- Superuser to close all database sessions
- None to close your own session

Examples

Two user sessions are open on separate nodes:
```sql
=> SELECT * FROM sessions;
- [ RECORD 1 ]---------------------------------------------------------------
  node_name   | v_vmartdb_node0001
  user_name   | dbadmin
  client_hostname | 127.0.0.1:52110
  client_pid  | 4554
  login_timestamp | 2011-01-03 14:05:40.252625-05
  session_id  | stress04-4325:0x14
  client_label | |
  transaction_start | 2011-01-03 14:05:44.325781
  transaction_id  | 45035996273728326
  transaction_description | user dbadmin (select * from sessions);
  statement_start | 2011-01-03 15:36:13.896288
  statement_id  | 10
  last_statement_duration_us | 14978
  current_statement | select * from sessions;
  ssl_state  | None
  authentication_method | Trust
- [ RECORD 2 ]---------------------------------------------------------------
  node_name   | v_vmartdb_node0002
  user_name   | dbadmin
  client_hostname | 127.0.0.1:57174
  client_pid  | 30117
  login_timestamp | 2011-01-03 15:33:00.842021-05
  session_id  | stress05-27944:0xc1a
  client_label | |
  transaction_start | 2011-01-03 15:34:46.538102
  transaction_id  | -1
  transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/mart_Fact.tbl'
                        DELIMITER '|' NULL '\n';)
  statement_start | 2011-01-03 15:34:46.538862
  statement_id  | |
  last_statement_duration_us | 26250
  current_statement | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|
                     NULL '\n';
  ssl_state  | None
  authentication_method | Trust
- [ RECORD 3 ]---------------------------------------------------------------
  node_name   | v_vmartdb_node0003
  user_name   | dbadmin
  client_hostname | 127.0.0.1:56367
  client_pid  | 1191
  login_timestamp | 2011-01-03 15:31:44.93902-05
  session_id  | stress06-25663:0xbec
  client_label | |
  transaction_start | 2011-01-03 15:34:51.05939
  transaction_id  | 54043195528458775
  transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/Mart_Fact.tbl'
                        DELIMITER '|' NULL '\n' DIRECT;)
  statement_start | 2011-01-03 15:34:51.05939
  statement_id  | |
  last_statement_duration_us | 1591403
  current_statement | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|
                     NULL '\n' DIRECT;
  ssl_state  | None
  authentication_method | Trust
```

Close all sessions:
Expanded display is off.

`SELECT CLOSE_ALL_SESSIONS();
CLOSE_ALL_SESSIONS
------------------------------------------`

Close all sessions command sent. Check `v_monitor.sessions` for progress.
(1 row)

**Session contents after issuing CLOSE_ALL_SESSIONS:**

`=> SELECT * FROM SESSIONS;
- [ RECORD 1 ]------------------------------------------
nodename | v_vmartdb_node001
user_name | dbadmin
client_hostname | 127.0.0.1:52110
client_pid | 4554
login_timestamp | 2011-01-03 14:05:40.252625-05
session_id | stress04-4325:0x14
client_label | 
transaction_start | 2011-01-03 14:05:44.325781
transaction_id | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start | 2011-01-03 16:19:56.720071
statement_id | 25
last_statement_duration_us | 15605
current_statement | SELECT * FROM SESSIONS;
ssl_state | None
authentication_method | Trust`

See Also

- `< Cue Mark>` CLOSE_SESSION
- `< Cue Mark>` CLOSE_USER_SESSIONS
- `< Cue Mark>` SHUTDOWN

**CLOSE_SESSION**

Interrupts the specified external session, rolls back the current transaction if any, and closes the socket. You can only close your own session.

It might take some time before a session is closed. To view the status of all open sessions, query the system table `SESSIONS`.

For detailed information about session management options, see Managing Sessions in the Administrator's Guide.
Syntax

CLOSE_SESSION ( 'sessionid')

Parameters

| sessionid       | A string that specifies the session to close. This identifier is unique within the cluster at any point in time but can be reused when the session closes. |

Privileges

None

Examples

User session opened. Record 2 shows the user session running a COPY DIRECT statement.

```
=> SELECT * FROM sessions;
-[ RECORD 1 ]---------------------------
node_name       | v_vmartdb_node001
user_name       | dbadmin
client_hostname | 127.0.0.1:52110
client_pid      | 4554
login_timestamp | 2011-01-03 14:05:40.252625-05
session_id      | stress04-4325:0x14
client_label    | 
transaction_start | 2011-01-03 14:05:44.325781
transaction_id  | 45035996273728326
transaction_description | user dbadmin (SELECT * FROM sessions;)
statement_start | 2011-01-03 15:36:13.096288
statement_id    | 10
last_statement_duration_us | 14978
current_statement | select * from sessions;
ssl_state       | None
authentication_method | Trust

-[ RECORD 2 ]---------------------------
node_name       | v_vmartdb_node002
user_name       | dbadmin
client_hostname | 127.0.0.1:57174
client_pid      | 30117
login_timestamp | 2011-01-03 15:33:00.842021-05
session_id      | stress05-27944:0xc1a
client_label    | 
transaction_start | 2011-01-03 15:34:46.538102
transaction_id  | -1
transaction_description | user dbadmin (COPY ClickStream_Fact FROM '/data/clickstream/1g/ClickStream_Fact.tbl' DELIMITER '|' NULL '\n' DIRECT;)
statement_start | 2011-01-03 15:34:46.538862
```
Close user session stress05-27944:0xc1a

=> \x
Expanded display is off.
=> SELECT CLOSE_SESSION('stress05-27944:0xc1a');
CLOSE_SESSION
------------------------------------------
Session close command sent. Check v_monitor.sessions for progress.
(1 row)

Query the sessions table again for current status, and you can see that the second session has been closed:

=> SELECT * FROM SESSIONS;
-[ RECORD 1 ]------------------------------------------
node_name | v_vmartdb_node0001
user_name | dbadmin
client_hostname | 127.0.0.1:52110
client_pid | 4554
login_timestamp | 2011-01-03 14:05:40.252625-05
session_id | stress04-4325:0x14
client_label |
transaction_start | 2011-01-03 14:05:44.325781
transaction_id | 4503596273728326
transaction_description | user dbadmin (select * from SESSIONS;)
statement_start | 2011-01-03 16:12:07.841298
statement_id | 20
last_statement_duration_us | 2099
current_statement | SELECT * FROM SESSIONS;
ssl_state | None
authentication_method | Trust

See Also

- CLOSE_ALL_SESSIONS
- SHUTDOWN

CLOSE_USER_SESSIONS

Stops the session for a user, rolls back any transaction currently running, and closes the connection. To determine the status of the sessions to close, query the SESSIONS table.
### CLOSE_USER_SESSIONS

#### Syntax

```sql
CLOSE_USER_SESSIONS ( 'user-name' )
```

#### Parameters

| **user-name** | Specifies the user whose sessions are to be closed. If you specify your own user name, Vertica closes all sessions except the one in which you issue this function. |

#### Privileges

**DBADMIN**

#### Examples

This example closes all active session for user u1:

```sql
=> SELECT close_user_sessions('u1');
```

### See Also

- CLOSE_ALL_SESSIONS
- CLOSE_SESSION
- SHUTDOWN

### GET_NUM_ACCEPTED_ROWS

Returns the number of rows loaded into the database for the last completed load for the current session. **GET_NUM_ACCEPTED_ROWS** is a meta-function. Do not use it as a value in an INSERT query.

The number of accepted rows is not available for a load that is currently in process. Check the **LOAD_STREAMS** system table for its status.
This meta-function supports loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use GET_NUM_ACCEPTED_ROWS for multi-node loads.

Syntax

```
GET_NUM_ACCEPTED_ROWS();
```

Privileges

None

Note: The data regarding accepted rows from the last load during the current session does not persist, and is lost when you initiate a new load.

Examples

This examples shows the number of accepted rows from the vmart_load_data.sql meta-command.

```
=> \\i vmart_load_data.sql;
=> SELECT GET_NUM_ACCEPTED_ROWS ();
GET_NUM_ACCEPTED_ROWS
-----------------------
300000
(1 row)
```

See Also

- `GET_NUM_REJECTED_ROWS`

GET_NUM_REJECTED_ROWS

Returns the number of rows that were rejected during the last completed load for the current session. GET_NUM_REJECTED_ROWS is a meta-function. Do not use it as a value in an INSERT query.

Rejected row information is unavailable for a load that is currently running. The number of rejected rows is not available for a load that is currently in process. Check the LOAD_STREAMS system table for its status.

This meta-function supports loads from STDIN, COPY LOCAL from a Vertica client, or a single file on the initiator. You cannot use GET_NUM_REJECTED_ROWS for multi-node loads.
Syntax

GET_NUM_REJECTED_ROWS();

Privileges

None

Note: The data regarding rejected rows from the last load during the current session does not persist, and is dropped when you initiate a new load.

Examples

This example shows the number of rejected rows from the vmart_load_data.sql meta-command.

```
=> \i vmart_load_data.sql
=> SELECT GET_NUM_REJECTED_ROWS();
GET_NUM_REJECTED_ROWS
-----------------------
0
(1 row)
```

See Also

- GET_NUM_ACCEPTED_ROWS

**INTERRUPT_STATEMENT**

Interrupts the specified statement in a user session, rolls back the current transaction, and writes a success or failure message to the log file.

Sessions can be interrupted during statement execution. Only statements run by user sessions can be interrupted.

Syntax

INTERRUPT_STATEMENT( 'session-id', statement-id )
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session-id</td>
<td>Identifies the session to interrupt. This identifier is unique within the</td>
</tr>
<tr>
<td></td>
<td>cluster at any point in time.</td>
</tr>
<tr>
<td>statement-id</td>
<td>Identifies the statement to interrupt. If the statement-id is valid, the</td>
</tr>
<tr>
<td></td>
<td>statement can be interrupted and INTERRUPT_STATEMENT returns a success</td>
</tr>
<tr>
<td></td>
<td>message. Otherwise the system returns an error.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

Messages

The following list describes messages you might encounter:

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement interrupt sent. Check SESSIONS for progress.</td>
<td>This message indicates success.</td>
</tr>
<tr>
<td>Session &lt;id&gt; could not be successfully interrupted: session not found.</td>
<td>The session ID argument to the interrupt command does not match a running session.</td>
</tr>
<tr>
<td>Session &lt;id&gt; could not be successfully interrupted: statement not found.</td>
<td>The statement ID does not match (or no longer matches) the ID of a running statement (if any).</td>
</tr>
<tr>
<td>No interruptible statement running</td>
<td>The statement is DDL or otherwise non-interruptible.</td>
</tr>
<tr>
<td>Internal (system) sessions cannot be interrupted.</td>
<td>The session is internal, and only statements run by external</td>
</tr>
</tbody>
</table>
Message | Meaning
--- | ---
| | sessions can be interrupted.

Examples

Two user sessions are open. RECORD 1 shows user session running `SELECT FROM SESSION`, and RECORD 2 shows user session running `COPY DIRECT`:

```sql
=> SELECT * FROM SESSIONS;

[-[ RECORD 1 ]-----------------------------------------------
node_name | v_vmartdb_node0001
user_name | dbadmin
client_hostname | 127.0.0.1:52110
client_pid | 4554
login_timestamp | 2011-01-03 14:05:40.252625-05
session_id | stress04-4325:0x14
client_label | |
transaction_start | 2011-01-03 14:05:44.325781
transaction_id | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start | 2011-01-03 15:36:13.896288
statement_id | 10
last_statement_duration_us | 14978
current_statement | select * from sessions;
ssl_state | None
authentication_method | Trust

[-[ RECORD 2 ]-----------------------------------------------
node_name | v_vmartdb_node0003
user_name | dbadmin
client_hostname | 127.0.0.1:56367
client_pid | 1191
login_timestamp | 2011-01-03 15:31:44.939302-05
session_id | stress06-25663:0xbec
client_label | |
transaction_start | 2011-01-03 15:34:51.05939
transaction_id | 54043195528458775
transaction_description | user dbadmin (COPY Mart_Fact FROM '/data/Mart_Fact.tbl'
| | DELIMITER '|' NULL '\n' DIRECT;)
statement_start | 2011-01-03 15:35:46.436748
statement_id | 5
last_statement_duration_us | 1591403
current_statement | COPY Mart_Fact FROM '/data/Mart_Fact.tbl' DELIMITER '|' NULL '\n' DIRECT;
ssl_state | None
authentication_method | Trust
```

Interrupt the COPY DIRECT statement running in session `stress06-25663:0xbec`:

```sql
=> \x
Expanded display is off.
=> SELECT INTERRUPT_STATEMENT('stress06-25663:0x1537', 5);
```

Vertica Analytic Database (9.0.x)
Verify that the interrupted statement is no longer active by looking at the `current_statement` column in the `SESSIONS` system table. This column becomes blank when the statement is interrupted:

```sql
=> SELECT * FROM SESSIONS;

node_name          | v_vmartdb_node0001
user_name           | dbadmin
client_hostname     | 127.0.0.1:52110
client_pid          | 4554
login_timestamp     | 2011-01-03 14:05:40.252625-05
session_id          | stress04-4325:8x14
client_label        | transaction_start | 2011-01-03 14:05:44.325781
transaction_id      | 45035996273728326
transaction_description | user dbadmin (select * from sessions;)
statement_start     | 2011-01-03 15:36:14.896208
statement_id        | 10
last_statement_duration_us | 14978
current_statement   | select * from sessions;
ssl_state           | None
authentication_method | Trust
```

**See Also**

- [SESSIONS](#)
- Managing Sessions
- Configuration Parameters
RELEASE_ALL_JVM_MEMORY

Forces all sessions to release the memory consumed by their Java Virtual Machines (JVM).

Syntax

```
RELEASE_ALL_JVM_MEMORY();
```

Privileges

Must be a superuser.

Example

The following example demonstrates viewing the JVM memory use in all open sessions, then calling RELEASE_ALL_JVM_MEMORY() to release the memory:

```
=> select user_name, external_memory_kb FROM V_MONITOR_SESSIONS;
  user_name | external_memory_kb
-------------
  dbadmin    | 79705
(1 row)

=> SELECT RELEASE_ALL_JVM_MEMORY();
RELEASE_ALL_JVM_MEMORY

Close all JVM sessions command sent. Check v_monitor.sessions for progress.
(1 row)

=> SELECT user_name, external_memory_kb FROM V_MONITOR_SESSIONS;
  user_name | external_memory_kb
-------------
  dbadmin    | 0
(1 row)
```

See Also

- RELEASE_JVM_MEMORY

RELEASE_JVM_MEMORY

Terminates a Java Virtual Machine (JVM), making available the memory the JVM was using.
Syntax

RELEASE_JVM_MEMORY();

Privileges

None.

Examples

User session opened. RECORD 2 shows the user session running COPY DIRECT statement.

```
=> SELECT RELEASE_JVM_MEMORY();
    release_jvm_memory
-----------------------------
Java process killed and memory released
(1 row)
```

See Also

- RELEASE_ALL_JVM_MEMORY

RESERVE_SESSION_RESOURCE

Reserves memory resources from the general resource pool for the exclusive use of the Vertica backup and restore process. No other Vertica process can access reserved resources. If insufficient resources are available, Vertica queues the reservation request.

This metafunction is a session level reservation. When a session ends Vertica automatically releases any resources reserved in that session. Because the metafunction operates at the session level, the resource name does not need to be unique across multiple sessions.

You can view reserved resources by querying the SESSIONS table.

Syntax

RESERVE_SESSION_RESOURCE ( 'name', memory)
Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the resource to reserve.</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>The amount of memory in kilobytes to allocate to the resource.</td>
</tr>
</tbody>
</table>

Privileges

None

Example

Reserve 1024 kilobytes of memory for the backup and restore process:

```
=> SELECT reserve_session_resource('VBR_RESERVE',1024);
- [ RECORD 1 ]-----------------------------------------------
             reserve_session_resource | Grant succeed
```

RESET_SESSION

Applies your default connection string configuration settings to your current session.

Syntax

```
RESET_SESSION()
```

Examples

The following example shows how you use RESET_SESSION.

Resets the current client connection string to the default connection string settings:

```
=> SELECT RESET_SESSION();
  RESET_SESSION
------------------
  Reset session: done.
(1 row)
```
Statistic Management Functions

This section contains statistic management functions specific to Vertica.

ANALYZE_EXTERNAL_ROW_COUNT

Calculates the exact number of rows in an external table. ANALYZE_EXTERNAL_ROW_COUNT runs in the background.

**Note:** You cannot calculate row counts on external tables with \texttt{DO_TM_TASK}.

**Syntax**

\begin{verbatim}
ANALYZE_EXTERNAL_ROW_COUNT ('[[\{database\}\{.\}\{schema\}\{.\}\{table-name\} \}]')
\end{verbatim}

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{database}{.}\texttt{schema}</td>
<td>Specifies a schema, by default public. If \texttt{schema} is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>\texttt{table-name}</td>
<td>Specifies the name of the external table for which to calculate the exact row count. If you supply an empty string, Vertica calculate the exact number of rows for all external tables.</td>
</tr>
</tbody>
</table>

**Privileges**

Any \texttt{INSERT/UPDATE/DELETE} privilege on the external table

**Examples**

Calculate the exact row count for all external tables:

```
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('');
```

Calculate the exact row count for table \texttt{loader_rejects}:
=> SELECT ANALYZE_EXTERNAL_ROW_COUNT('loader_rejects');

See Also

- Collecting Database Statistics
- DROP_EXTERNAL_ROW_COUNT

ANALYZE_STATISTICS

Note: ANALYZE_STATISTICS is an alias of the function ANALYZE_HISTOGRAM, which is no longer documented.

Collects and aggregates data samples and storage information from all nodes that store projections associated with the specified table. By default, Vertica analyzes multiple columns in a single-query execution plan, depending on resource limits. Such multi-column analysis facilitates the following objectives:

- Reduce plan execution latency.
- Speed up analysis of relatively small tables with many columns.

Vertica writes statistics to the database catalog. The query optimizer uses this collected data to create query plans. Without this data, the query optimizer assumes uniform distribution of data values and equal storage usage for all projections.

You can cancel statistics collection with CTRL+C or by calling INTERRUPT_STATEMENT.

Syntax

ANALYZE_STATISTICS ('[ scope ]' [, 'column[,...]' ] [, percent ] )

Returns

0—Success

If an error occurs, refer to vertica.log for details.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>scope</strong></td>
<td>Specifies the table on which to collect data, as follows: <code>[[database.]schema.]table</code> If set to an empty string, Vertica collects statistics for all database tables and their projections. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><strong>column</strong></td>
<td>The name of a column in <em>table</em>, typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of the data collection to the specified columns. If you alter a table to add a column and populate its contents with either default or other values, call <code>ANALYZE_STATISTICS</code> on this column to get the most current statistics.</td>
</tr>
<tr>
<td><strong>percent</strong></td>
<td>A float value between 0 and 100 that specifies what percentage of data to read from disk (not the amount of data to analyze). If you omit this argument, Vertica sets the percentage to 10. Analyzing more than 10 percent disk space takes proportionally longer to process, but produces a higher level of sampling accuracy.</td>
</tr>
</tbody>
</table>

Privileges

- Any INSERT/UPDATE/DELETE privilege on the specified table.
- USAGE privilege on schema that contains the table

Restrictions

- Vertica collects no statistics on live aggregate and Top-K projections that are anchored to the specified table.
- If you include a SQL function within an expression when you create a projection, Vertica collects no statistics for that projection.
- Vertica supports `ANALYZE_STATISTICS` on local temporary tables but not on global temporary tables. To obtain statistics on a temporary table, create the table with the option...
ON COMMIT PRESERVE ROWS. Otherwise, Vertica deletes the table content when it commits the current transaction, so no table data is available for analysis.

See Also

- Getting Statistics
- DROP_STATISTICS
- EXPORT_STATISTICS
- IMPORT_STATISTICS
- VALIDATE_STATISTICS

DROP_EXTERNAL_ROW_COUNT

Removes external table row count statistics compiled by ANALYZEEXTERNAL_ROW_COUNT. DROPEXTERNAL_ROW_COUNT runs in the background.

Caution: Statistics can be time consuming to regenerate.

Syntax

DROP_EXTERNAL_ROW_COUNT ('[[[database.]schema.]table-name ]');

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>schema</strong></td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><strong>table-name</strong></td>
<td>The external table for which to remove the exact row count. If you specify an empty string, Vertica drops the exact row count statistic for all external tables.</td>
</tr>
</tbody>
</table>
Privileges

- INSERT/UPDATE/DELETE privilege on table
- USAGE privilege on schema that contains the table

Examples

Drop row count statistics for external table `loader_rejects`:

```sql
=> SELECT DROP_EXTERNAL_ROW_COUNT('loader_rejects');
```

See Also

Collecting Database Statistics

DROP_STATISTICS

Removes statistical data on database projections previously generated by `ANALYZE_STATISTICS`. When you drop this data, the Vertica optimizer uses default statistics to create query plans.

**Caution:** Before you drop statistics, be aware that statistics can be time consuming to regenerate.

Syntax

`DROP_STATISTICS ('[ scope ][, 'category'][ , 'column[,...]'] )`

Parameters

<table>
<thead>
<tr>
<th><strong>scope</strong></th>
<th>Specifies the target table as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>[[database.]schema.]table</code></td>
</tr>
<tr>
<td></td>
<td>If set to an empty string, Vertica drops statistics for projections of all database tables.</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><strong>category</strong></td>
<td>The category of statistics to drop for the named table, one of the</td>
</tr>
</tbody>
</table>
following:

- BASE (default) drops histograms and row counts (min/max column values, histogram).
- HISTOGRAMS drops only histograms. Row counts statistics remain.
- ALL drops all statistics.

| column | The name of a column in table, typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of dropped statistics to the specified columns only. |

Privileges

- INSERT/UPDATE/DELETE privilege on table
- USAGE privilege on schema that contains the table

Examples

Drop all base statistics for the table store.store_sales_fact:

```
=> SELECT DROP_STATISTICS('store.store_sales_fact');
DROP_STATISTICS
------------------
  0
(1 row)
```

Drop statistics for all table projections:

```
=> SELECT DROP_STATISTICS ('');
DROP_STATISTICS
------------------
  0
(1 row)
```

See Also

- ANALYZE_STATISTICS
- EXPORT_STATISTICS
- **IMPORT_STATISTICS**
- **VALIDATE_STATISTICS**

**EXPORT_STATISTICS**

Generates database statistics in XML format from data previously collected by `ANALYZE_STATISTICS`. Before you export statistics, collect the latest data by calling `ANALYZE_STATISTICS`.

**Syntax**

`EXPORT_STATISTICS ('[filename][, scope][, column[,...]]')`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Specifies where to write the generated XML. If <code>filename</code> already exists, <code>EXPORT_STATISTICS</code> overwrites it. If you supply an empty string, <code>EXPORT_STATISTICS</code> writes the XML to standard output.</td>
</tr>
</tbody>
</table>
| `scope` | Specifies the target table as follows:  
`[[database.]schema.]table`  
If you specify a table, Vertica exports statistics for its projections. If you supply an empty string, Vertica exports all statistics for the database.  
If you specify a database, it must be the current database. |
| `column` | The name of a column in `table`, typically a predicate column. You can specify multiple comma-delimited columns. Vertica narrows the scope of exported statistics to the specified columns only. |

**Privileges**

Superuser

**Restrictions**

`EXPORT_STATISTICS` does not export statistics for LONG data type columns.
Examples

The following statement exports statistics on the VMart example database to a file:

```sql
=> SELECT EXPORT_STATISTICS('/opt/vertica/examples/VMart_Schema/vmart_stats.xml');
EXPORT_STATISTICS
------------------------------------------
Statistics exported successfully
(1 row)
```

The next statement exports statistics on a single column (price) from a table named food:

```sql
=> SELECT EXPORT_STATISTICS('/opt/vertica/examples/VMart_Schema/price.xml', 'food.price');
EXPORT_STATISTICS
------------------------------------------
Statistics exported successfully
(1 row)
```

See Also

- `ANALYZE_STATISTICS`
- `DROP_STATISTICS`
- `IMPORT_STATISTICS`
- `VALIDATE_STATISTICS`
- Collecting Database Statistics
- Best Practices For Statistics Collection

**IMPORT_STATISTICS**

Imports statistics from the XML file that was generated by `EXPORT_STATISTICS`. Imported statistics override existing statistics for all projections of the table referenced in the XML file.

**Syntax**

```sql
IMPORT_STATISTICS ('filename')
```
Parameters

| filename | The path and name of an XML input file that was generated by \texttt{EXPORT\_STATISTICS}. |

Privileges

Superuser

Restrictions

- \texttt{IMPORT\_STATISTICS} imports only valid statistics. If the source XML file has invalid statistics for a specific column, those statistics are not imported and Vertica throws a warning. If the statistics file has an invalid structure, the import operation fails. To check a statistics file for validity, run \texttt{VALIDATE\_STATISTICS}.

- \texttt{IMPORT\_STATISTICS} returns warnings for LONG data type columns, as the source XML file generated by \texttt{EXPORT\_STATISTICS} contains no statistics for columns of that type.

Example

Import the statistics for the VMart database that \texttt{EXPORT\_STATISTICS} saved.

```sql
=> SELECT IMPORT\_STATISTICS('/opt/vertica/examples/VMart_\_Schema/vmart_stats.xml');

IMPORT\_STATISTICS

-------------------------------------------------------------
Importing statistics for projection date\_dimension\_super column date\_key failure (stats did not contain row counts)
Importing statistics for projection date\_dimension\_super column date failure (stats did not contain row counts)
Importing statistics for projection date\_dimension\_super column full\_date\_description failure (stats did not contain row counts)
...
(1 row)
```

See Also

- \texttt{ANALYZE\_STATISTICS}
- \texttt{DROP\_STATISTICS}
**EXPORT_STATISTICS**

- Collecting Statistics in the Administrator's Guide

**VALIDATE_STATISTICS**

Validates statistics in the XML file generated by the EXPORT_STATISTICS command.

**Syntax**

```sql
VALIDATE_STATISTICS ( 'XML_statistics_file' )
```

**Parameters**

<table>
<thead>
<tr>
<th>XML_statistics_file</th>
<th>Specifies the path and name of the XML file that contains the statistics you want to validate, type VARCHAR.</th>
</tr>
</thead>
</table>

**Privileges**

Superuser

**Usage Considerations**

- **IMPORT_STATISTICS** imports only valid statistics. If the source file has invalid statistics for a specific column, those statistics are not imported and a warning occurs. To identify the invalid statistics, run VALIDATE_STATISTICS.

**Example: Valid Statistics**

The following example shows the results when the statistics are valid:

```sql
=> SELECT EXPORT_STATISTICS('cust_dim_stats.xml','customer_dimension');
EXPORT_STATISTICS
--------------------------
Statistics exported successfully
(1 row)
=> SELECT VALIDATE_STATISTICS('cust_dim_stats.xml');
VALIDATE_STATISTICS
----------------------
(1 row)
```
Example: Invalid Statistics File

The following example shows the results when some of the statistics are invalid. For example, the 'distinct', 'buckets', 'rows', 'count', and 'distinctCount' attributes cannot be negative numbers. Vertica recommends that you rerun `ANALYZE_STATISTICS` on this table to create valid statistics:

```sql
=> SELECT VALIDATE_STATISTICS('/stats.xml');
WARNING 0: Invalid value '-1' for attribute 'distinct' under column 'public.t.x'.
  Please use a positive value.
WARNING 0: Invalid value '-1' for attribute 'buckets' under column 'public.t.x'.
  Please use a positive value.
WARNING 0: Invalid value '-1' for attribute 'rows' under column 'public.t.x'.
  Please use a positive value.
WARNING 0: Invalid value '-1' for attribute 'count' under bound '1', column 'public.t.x'.
  Please use a positive value.
WARNING 0: Invalid value '-1' for attribute 'distinctCount' under bound '1', column 'public.t.x'.
  Please use a positive value.
VALIDATE_STATISTICS
```

Example: Invalid Statistics File

The following example shows the result when the statistics file is invalid:

```sql
=> SELECT VALIDATE_STATISTICS('/home/dbadmin/stats.xml');
    
VALIDATE_STATISTICS
```

Error validating statistics file: At line 1:1. Invalid document structure

(1 row)

See Also

- `ANALYZE_STATISTICS`
- `DROP_STATISTICS`
- `EXPORT_STATISTICS`
- `IMPORT_STATISTICS`
Storage Management Functions

This section contains storage management functions specific to Vertica.

**ALTER_LOCATION_LABEL**

Alters the location label. Use this function to add, change, or remove a location label. You change a location label only if it is not currently in use as part of a storage policy.

You can use this function to remove a location label. However, you cannot remove a location label if the name being removed is used in a storage policy, and the location from which you are removing the label is the last available storage for its associated objects.

*Note:* If you label an existing storage location that already contains data, and then include the labeled location in one or more storage policies, existing data could be moved. If the ATM determines data stored on a labeled location does not comply with a storage policy, the ATM moves the data elsewhere.

**Syntax**

```
ALTER_LOCATION_LABEL ('path', 'node', 'location_label')
```

**Parameters**

- **path**
  Specifies the path of the storage location.

- **node**
  The Vertica node for the storage location.
  If you enter node as an empty string (''), the function performs a cluster-wide label change to all nodes. Any node that is unavailable generates an error.

- **location_label**
  Specifies a storage label as a string, for instance SSD. You can change an existing label assigned to a storage location, or add a new label. Specifying an empty string ('') removes an existing label.

**Privileges**

Superuser
Example

The following example alters (or adds) the label SSD to the storage location at the given path on all cluster nodes:

```sql
=> SELECT alter_location_label('/home/dbadmin/SSD/tables','','SSD');
alter_location_label
------------------------------------------
/home/dbadmin/SSD/tables label changed.
(1 row)
```

See Also

- Altering Location Labels
- CLEAR_OBJECT_STORAGE_POLICY
- SET_OBJECT_STORAGE_POLICY

ALTER_LOCATION_USE

Alters the type of files that can be stored at the specified storage location.

Syntax

```sql
ALTER_LOCATION_USE ( 'path' , 'node' , 'usage' )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>path</code></td>
<td>Specifies where the storage location is mounted.</td>
</tr>
<tr>
<td><code>node</code></td>
<td>[Optional] The Vertica node with the storage location. Specifying the <code>node</code> parameter as an empty string (' ') alters the location across all cluster nodes in a single transaction.</td>
</tr>
<tr>
<td><code>usage</code></td>
<td>Is one of the following:</td>
</tr>
<tr>
<td></td>
<td>- DATA: The storage location stores only data files. This is the supported use for both a USER storage location, and a labeled storage location.</td>
</tr>
<tr>
<td></td>
<td>- TEMP: The location stores only temporary files that are created during loads or</td>
</tr>
</tbody>
</table>
queries.

- DATA, TEMP: The location can store both types of files.

Privileges

Superuser

USER Storage Location Restrictions

You cannot change a storage location from a USER usage type if you created the location that way, or to a USER type if you did not. You can change a USER storage location to specify DATA (storing TEMP files is not supported). However, doing so does not affect the primary objective of a USER storage location, to be accessible by non-dbadmin users with assigned privileges.

Monitoring Storage Locations

Disk storage information that the database uses on each node appears in the V_MONITOR.DISK_STORAGe system table.

Example

The following example alters the storage location across all cluster nodes to store only data:

```sql
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/', '', 'DATA');
```

See Also

- Altering Location Use
- DROP_LOCATION
- RESTORE_LOCATION
- RETIRE_LOCATION
- GRANT (Storage Location)
- REVOKE (Storage Location)
CLEAR_CACHES

Clears the Vertica internal cache files.

Syntax

CLEAR_CACHES ( )

 Privileges

Superuser

Notes

If you want to run benchmark tests for your queries, in addition to clearing the internal Vertica cache files, clear the Linux file system cache. The kernel uses unallocated memory as a cache to hold clean disk blocks. If you are running version 2.6.16 or later of Linux and you have root access, you can clear the kernel filesystem cache as follows:

1. Make sure that all data in the cache is written to disk:

   # sync

2. Writing to the drop_caches file causes the kernel to drop clean caches, entries, and inodes from memory, causing that memory to become free, as follows:

   - To clear the page cache:

     # echo 1 > /proc/sys/vm/drop_caches

   - To clear the entries and inodes:

     # echo 2 > /proc/sys/vm/drop_caches

   - To clear the page cache, entries, and inodes:

     # echo 3 > /proc/sys/vm/drop_caches
Example

The following example clears the Vertica internal cache files:

```sql
=> SELECT CLEAR_CACHES();
CLEAR_CACHES
------------
Cleared
(1 row)
```

**CLEAR_OBJECT_STORAGE_POLICY**

Removes an existing storage policy. The specified object will no longer use a user-created storage location. Any existing data stored currently at the labeled location in the object's storage policy is moved to default storage during the next TM moveout operation.

**Syntax**

```sql
CLEAR_OBJECT_STORAGE_POLICY ( '[database.]schema.object-name' [, 'key-min', 'key-max'] [, 'enforce-storage-move' ]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td><code>object-name</code></td>
<td>Identifies the database object whose storage policies are to be cleared. The <code>object_name</code> parameter can resolve to a database, schema, or table.</td>
</tr>
<tr>
<td><code>key-min</code> <code>key-max</code></td>
<td>Specifies the table partition key value ranges stored at the labeled location. These parameters are applicable only when <code>object_name</code> is a table.</td>
</tr>
<tr>
<td><code>enforce-storage-move</code></td>
<td>If <code>true</code>, moves the storage containers that belong to this object to the new location immediately. If <code>false</code> (the default), this move occurs automatically sometime after the</td>
</tr>
</tbody>
</table>
Privileges

Superuser

Examples

This example shows how to clear the storage policy for the object `lineorder`. Changes take effect after the next moveout:

```sql
=> select clear_object_storage_policy('lineorder');
       clear_object_storage_policy
----------------------------
 Default storage policy cleared.
(1 row)
```

See Also

- Clearing Storage Policies
- `ALTER_LOCATION_LABEL`
- `SET_OBJECT_STORAGE_POLICY`
- `ENFORCE_OBJECT_STORAGE_POLICY`

COMPACT_STORAGE

Bundles existing data (.fdb) and index (.pidx) files into the .gt file format. The .gt format was introduced in Vertica 7.2 and is enabled by default for any data files created version 7.2 or later. If you have a database from an earlier version, you can use COMPACT_STORAGET to upgrade existing storage files to the new format. Your database can continue to operate with a mix of file storage formats.

If the settings you specify for COMPACT_STORAGE vary from the limit specified in configuration parameter `MaxBundleableROSSizeKB`, Vertica does not change the size of the automatically created bundles. You can use this function even if storage bundling is not enabled on your database.

Note: Run this function during periods of low demand.
Syntax

SELECT COMPACT_STORAGE ('[[database.]schema.]object-name', min-ros-filesize-kb, 'small-or-all-files', 'simulate');

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>object-name</td>
<td>Specifies the table or projection to bundle.</td>
</tr>
<tr>
<td>min-ros-filesize-kb</td>
<td>Specifies the minimum size, in kilobytes, of an independent ROS file. Vertica bundles storage container ROS files below this size into a single file. If set to 0, Vertica bundles the data and index files of an individual column, but not with other columns in that storage container.</td>
</tr>
<tr>
<td>small-or-all-files</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>small: Bundles only files smaller than the limit specified in min-ros-filesize-kb</td>
</tr>
<tr>
<td></td>
<td>all: Bundles files smaller than the limit specified in min-ros-filesize-kb and bundles the .fdb and .pidx files for larger storage containers.</td>
</tr>
<tr>
<td>simulate</td>
<td>Specifies whether to simulate the storage settings and produce a report describing the impact of those settings.</td>
</tr>
<tr>
<td></td>
<td>true: Produces a report on the impact of the specified bundle settings without actually bundling storage files.</td>
</tr>
<tr>
<td></td>
<td>false: Vertica performs the bundling according to the settings you specified.</td>
</tr>
</tbody>
</table>
Privileges
Superuser

Storage and Performance Impact

Bundling reduces the number of files in your file system by at least fifty percent and improves the performance of file-intensive operations. Improved operations include backups, restores, mergeouts and moveouts.

Vertica creates small files for the following reasons:

- Tables contain hundreds of columns.
- Partition ranges are small (partition by minute).
- Local segmentation is enabled and your factor is set to a high value.

Evaluating the Benefits of Bundled Storage

You can determine whether bundling existing storage can provide a significant benefit, as follows:

**View the median file size of a projection on a node**

Run the following query:

```sql
SELECT MEDIAN(size) OVER() AS median_fsize
FROM vs_ros AS ros, storage_containers AS cont
WHERE ros.delid=cont.storage_oid
AND cont.node_name='node-name'
AND cont.projection_name='proj-name' LIMIT 1;
```

If many files are smaller than 1mb, bundling storage is likely to provide significant performance benefits.

**Run COMPACT_STORAGE in simulation mode**

The simulate parameter produces a report that shows the number of affected files and the change in file storage.

**Example**

The following example describes the impact of bundling the table EMPLOYEES:
DROP_LOCATION

Removes the specified storage location. Dropping a storage location is a permanent operation and cannot be undone. Therefore, you must retire a storage location before dropping it.

Syntax

DROP_LOCATION ( 'path', 'node' )

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Specifies where the storage location to drop is mounted.</td>
</tr>
<tr>
<td>node</td>
<td>The Vertica node where the location is available. A value of &quot; (an empty string) means to perform this operation on all nodes.</td>
</tr>
</tbody>
</table>
Privileges
Superuser

Retiring a Storage Location
Retiring a storage location lets you verify that you do not need the storage before dropping it. You can also restore a retired storage location if you determine it is still in use.

Dropping a Shared Location
When a storage location is created in a shared location, such as HDFS, Vertica creates subdirectories for each node to prevent conflicts. For example, suppose you create a shared storage location in /data. The location for v_vmart_node0001 is /data/v_vmart_node0001, the location for v_vmart_node0002 is /data/v_vmart_node0002, and so on. To drop a location on only one node, use the node-specific path such as /data/v_vmart_node0002. To drop a location on all nodes, use the same path that you used to create it (/data in this example).

Storage Locations with Temp and Data Files
If you use a storage location to store data and then alter it to store only temp files, the location can still contain data files. Vertica does not let you drop a storage location containing data files. You can use the function MOVE_RETIRED_LOCATION_DATA to manually merge out the data files from the storage location, or you can drop partitions. Deleting data files does not work.

Examples
The following example shows how to drop a previously retired storage location on v_vmart_node0003:

```sql
=> SELECT DROP_LOCATION('/data', 'v_vmart_node003');
```

See Also
- Dropping Storage Locations
- Retiring Storage Locations
- ALTER_LOCATION_USE
• **RESTORE_LOCATION**

• **RETIRE_LOCATION**

• **GRANT (Storage Location)**

• **REVOKE (Storage Location)**

**ENFORCE_OBJECT_STORAGE_POLICY**

Applies object storage policies immediately, instead of waiting for the Tuple Mover to perform the next moveout. Calling this function is equivalent to setting the `enforce_storage_move` parameter on related meta-functions. You typically use this function as the last step before dropping a storage location.

This function invokes the Tuple Mover on a one-time basis.

**Syntax**

```
ENFORCE_OBJECT_STORAGE_POLICY ( 'object_name', [', 'key_min', ', 'key_max'] )
```

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object_name</code></td>
<td>Identifies the database object whose storage policies are to be applied. The <code>object_name</code> parameter can resolve to a database, schema, or table.</td>
</tr>
<tr>
<td><code>key_min, key_max</code></td>
<td>Applicable only when <code>object_name</code> is a table, <code>key_min</code> and <code>key_max</code> specify the table partition key value range over which to perform the moves.</td>
</tr>
</tbody>
</table>

**Privileges**

Must be the object owner to enforce the storage policy and have access to the storage location.

**Examples**

This example shows how to apply storage-policy updates to the `test` table:

```
=> select enforce_object_storage_policy('test');
```
See Also

- **CLEAR_OBJECT_STORAGE_POLICY**
- **RETIRE_LOCATION**
- **DROP_LOCATION**
- Managing Storage Locations

**MEASURE_LOCATION_PERFORMANCE**

Measures disk performance for the location specified.

**Syntax**

```sql
MEASURE_LOCATION_PERFORMANCE ( 'path', 'node' )
```

**Parameters**

<table>
<thead>
<tr>
<th>path</th>
<th>Specifies where the storage location to measure is mounted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>Is the Vertica node where the location to be measured is available.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Notes**

- To get a list of all node names on your cluster, query the `V_MONITOR.DISK_STORAGE` system table:

```sql
=> SELECT node_name from DISK_STORAGE;
node_name
---------
v_vmartdb_node0004
v_vmartdb_node0004
v_vmartdb_node0005
```
If you intend to create a tiered disk architecture in which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns, you need to measure storage location performance for each location in which data is stored. You do not need to measure storage location performance for temp data storage locations because temporary files are stored based on available space.

The method of measuring storage location performance applies only to configured clusters. If you want to measure a disk before configuring a cluster see Measuring Storage Performance.

Storage location performance equates to the amount of time it takes to read and write 1MB of data from the disk. This time equates to:

\[ \text{IO time} = \text{Time to read/write 1MB} + \text{Time to seek} = 1/\text{Throughput} + 1/\text{Latency} \]

Throughput is the average throughput of sequential reads/writes (units in MB per second)

Latency is for random reads only in seeks (units in seeks per second)

**Note:** The IO time of a faster storage location is less than a slower storage location.

**Example**

The following example measures the performance of a storage location on v_vmartdb_node0004:

```sql
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'v_vmartdb_node0004');
WARNING: measure_location_performance can take a long time. Please check logs for progress
                        measure_location_performance
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

**See Also**

- **CREATE LOCATION**
- **ALTER_LOCATION_USE**
• **RESTORE_LOCATION**

• **RETIRE_LOCATION**

• **Measuring Storage Performance**

**MOVE RETIRED LOCATION DATA**

Moves all data from either a single retired storage location or all retired storage locations in the database. This function migrates the data to non-retired storage locations based on the storage policies of the objects whose data is stored in the location.

**Syntax**

```sql
MOVE RETIRED LOCATION DATA(['location_path'] ['node'])
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>['location_path']</td>
<td>The path of the storage location as listed in the LOCATION_PATH column of the STORAGE_LOCATIONS system table. You must have previously marked this storage location as retired.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> If you do not supply a storage location path, this function moves data from all retired storage locations.</td>
</tr>
<tr>
<td>['node']</td>
<td>A specific node on which to move the retired storage location's data.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> If you do not specify a node, all nodes in the cluster move data from the retired location or locations. If a node does not define the storage location, this function returns an error.</td>
</tr>
</tbody>
</table>

**Privileges**

The user must be a superuser.

**Usage Considerations**

This function forces the Tuple Mover to move data out of the retired location or locations. Normally, the Tuple Mover only migrates data out of retired storage locations as it consolidates data into larger ROS containers. This function does not return until all of the data has moved off of the retired storage location or locations.
Examples

The following example:

1. Queries the STORAGE_LOCATIONS system table to show which storage locations are retired.

2. Queries the STORAGE_CONTAINERS system table to show the current location of the messages table, which is currently stored in the retired storage location named ssd.

3. Calls MOVE_RETIREDLOCATION_DATA to move the data off of the ssd storage location.

4. Repeats the previous query to show the storage location of the messages table.

```sql
=> SELECT node_name, location_path, location_label, is_retired FROM STORAGE_LOCATIONS
   WHERE is_retired = 't';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>location_path</th>
<th>location_label</th>
<th>is_retired</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>/home/dbadmin/SSDLoc</td>
<td>ssd</td>
<td>t</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>/home/dbadmin/SSDLoc</td>
<td>ssd</td>
<td>t</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>/home/dbadmin/SSDLoc</td>
<td>ssd</td>
<td>t</td>
</tr>
</tbody>
</table>

(3 rows)

```sql
=> SELECT node_name, total_row_count, storage_type, location_label FROM STORAGE_CONTAINERS
   WHERE projection_name ILIKE 'messages%';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>total_row_count</th>
<th>storage_type</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>333514</td>
<td>ROS</td>
<td>ssd</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>333255</td>
<td>ROS</td>
<td>ssd</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>333255</td>
<td>ROS</td>
<td>ssd</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>333231</td>
<td>ROS</td>
<td>ssd</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>333231</td>
<td>ROS</td>
<td>ssd</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>333514</td>
<td>ROS</td>
<td>ssd</td>
</tr>
</tbody>
</table>

(6 rows)

```sql
=> SELECT MOVE_RETIREDLOCATION_DATA('/home/dbadmin/SSDLoc');
```

MOVE_RETIREDLOCATION_DATA

Move data off retired storage locations done

(1 row)

```sql
=> SELECT node_name, total_row_count, storage_type, location_label FROM storage_containers
   WHERE projection_name ILIKE 'messages%';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>total_row_count</th>
<th>storage_type</th>
<th>location_label</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>333255</td>
<td>ROS</td>
<td>base</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>333514</td>
<td>ROS</td>
<td>base</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>333514</td>
<td>ROS</td>
<td>base</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>333231</td>
<td>ROS</td>
<td>base</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>333231</td>
<td>ROS</td>
<td>base</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>333255</td>
<td>ROS</td>
<td>base</td>
</tr>
</tbody>
</table>

(6 rows)
See Also

- `RETIRE_LOCATION`
- `RESTORE_LOCATION`
- `Managing Storage Locations` in the Administrator's Guide.

**RESTORE_LOCATION**

Restores a storage location that was previously retired with `RETIRE_LOCATION`.

**Syntax**

```sql
RESTORE_LOCATION ( 'path', 'node' )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>path</code></td>
<td>Specifies where the retired storage location is mounted.</td>
</tr>
<tr>
<td><code>node</code></td>
<td>The Vertica node where the retired location is available. A value of '' (an empty string) means to perform this operation on all nodes. The operation fails if you have dropped any locations.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Effects of Restoring a Previously Retired Location**

After restoring a storage location, Vertica re-ranks all of the cluster storage locations. It uses the newly restored location to process queries as determined by its rank.

**Monitoring Storage Locations**

Disk storage information that the database uses on each node appears in the `V_MONITOR.DISK_STORAGE` system table.
Examples

The following example shows how to restore the retired storage location on node3:

```
=> SELECT RESTORE_LOCATION ('/thirdVerticaStorageLocation/', 'v_vmartdb_node0004');
```

See Also

- Altering Location Use
- CREATE LOCATION
- ALTER_LOCATION_USE
- DROP_LOCATION
- RETIRE_LOCATION
- GRANT (Storage Location)
- REVOKE (Storage Location)

**RETIRE_LOCATION**

Makes the specified storage location inactive. To see a list of all of the storage locations in Vertica, refer to **STORAGE_LOCATIONS**.

Syntax

```
RETIRE_LOCATION ( 'path', 'node' [, enforce_storage_move ] )
```

Parameters

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Specifies where the storage location to retire resides.</td>
</tr>
<tr>
<td>node</td>
<td>The Vertica node where the location is available. A value of &quot;&quot; (an empty string) means to perform this operation on all nodes.</td>
</tr>
<tr>
<td>enforce_storage_move</td>
<td>[Optional] Use this to expedite dropping a location. If you set this argument to true, the location label is set to the empty string, and</td>
</tr>
</tbody>
</table>
the data is moved elsewhere. The location can then be dropped without errors or warnings.

Privileges
Superuser

Retiring a Shared Location

When you create a storage location in a shared HDFS location, Vertica creates subdirectories for each node to prevent conflicts. For example, suppose you create a shared storage location in /data. The location for v_vmartdb_node0001 is /data/v_vmartdb_node0001, the location for v_vmartdb_node0002 is /data/v_vmartdb_node0002, and so on. To retire a location on only one node, use the node-specific path such as /data/v_vmartdb_node0002. To retire a location on all nodes, use the same path that you used to create it (/data in this example).

Note: Vertica does not identify NFS as a shared file system.

Effects of Retiring a Storage Location

When you use this function, Vertica checks that the location is not the only storage for data and temp files. At least one location must exist on each node to store data and temp files. However, you can store both sorts of files in either the same location or separate locations.

Note: If a location is the last available storage for its associated objects, you cannot retire it unless you set enforce_storage_move to true

When you retire a storage location:

- No new data is stored at the retired location, unless you first restore it with the RESTORE_LOCATION() function.

- By default, if the storage location being retired contains stored data, the data is not moved. Thus, you cannot drop the storage location. Instead, Vertica removes the stored data through one or more mergeouts. If you want to drop the location immediately after retiring it, add the enforce_storage_move parameter, using a value of true.

- If the storage location being retired is used only for temp files or you use enforce_storage_move, you can drop the location. See Dropping Storage Locations in the Administrators Guide and the DROP_LOCATION() function.
Monitoring Storage Locations

Disk storage information that the database uses on each node appears in the V_MONITOR.DISK_STORAGE system table.

Examples

The following examples show two approaches to retiring a storage location.

You can specify that a storage location be dropped automatically at a future time:

```
=> SELECT RETIRE_LOCATION ('/data', 'v_vmartdb_node0004');
```

You can also specify that a storage location be dropped immediately:

```
=> SELECT RETIRE_LOCATION ('/data', 'v_vmartdb_node0004', true);
```

See Also

- Retiring Storage Locations
- CREATE LOCATION
- ALTER_LOCATION_USE
- DROP_LOCATION
- RESTORE_LOCATION
- GRANT (Storage Location)
- REVOKE (Storage Location)

SET_LOCATION_PERFORMANCE

Sets disk performance for the location specified.

Syntax

```
SET_LOCATION_PERFORMANCE ('path', 'node', 'throughput', 'average_latency')
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Specifies where the storage location to set is mounted.</td>
</tr>
<tr>
<td>node</td>
<td>Is the Vertica node where the location to be set is available. If this parameter is omitted, node defaults to the initiator.</td>
</tr>
<tr>
<td>throughput</td>
<td>Specifies the throughput for the location, which must be 1 or more.</td>
</tr>
<tr>
<td>average_latency</td>
<td>Specifies the average latency for the location. The average_latency must be 1 or more.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

Notes

To obtain the throughput and average latency for the location, run the MEASURE_LOCATION_PERFORMANCE() function before you attempt to set the location's performance.

Example

The following example sets the performance of a storage location on node2 to a throughput of 122 megabytes per second and a latency of 140 seeks per second.

```sql
=> SELECT SET_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/','node2','122','140');
```

See Also

- CREATE LOCATION
- MEASURE_LOCATION_PERFORMANCE
- Measuring Storage Performance
- Setting Storage Performance
SET_OBJECT_STORAGE_POLICY

Creates or changes an object storage policy by associating a database object with a labeled storage location.

Note: You cannot create a storage policy on a USER type storage location.

Syntax

```
SET_OBJECT_STORAGE_POLICY (
    '[[database.]schema.]object-name', 'location-Label'
    [, 'key-min', 'key-max'] [, 'enforce-storage-move']
)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>object-name</td>
<td>Identifies the database object assigned to a labeled storage location. The <code>object-name</code> can resolve to a database, schema, or table.</td>
</tr>
<tr>
<td>location-Label</td>
<td>The label of the storage location with which <code>object-name</code> is being associated.</td>
</tr>
<tr>
<td>key-min</td>
<td>Applicable only when <code>object-name</code> is a table, <code>key-min</code> and <code>key-max</code> specify the table partition key value range to be stored at the location.</td>
</tr>
<tr>
<td>key-max</td>
<td></td>
</tr>
<tr>
<td>enforce-storage-move</td>
<td>Specify this parameter as true to move all existing storage data to the target location within this function’s transaction. By default, data move the next time the Tuple Mover runs on the affected data.</td>
</tr>
</tbody>
</table>

Privileges

Must be the object owner to set the storage policy and have access to the storage location.
New Storage Policy

If an object does not have a storage policy, this function creates a new policy. The labeled location is then used as the default storage location during TM operations, such as moveout and mergeout.

Existing Storage Policy

If the object already has an active storage policy, calling this function changes the default storage for the object to the new labeled location. Any existing data stored on the previous storage location is marked to move to the new location during the next TM moveout operations. To move the data immediately, use the `enforce-storage-move` argument.

Forcing Existing Data Storage to a New Storage Location

You can optionally use this function to move existing data storage to a new location as part of completing the current transaction. To do so, set parameter `enforce-storage-move` to `true`. You might want to force a move, even though it means waiting for the operation to complete before continuing, if the data being moved is old. The Tuple Mover runs less frequently on older data.

To move existing data as part of the next TM moveout, either omit the parameter, or specify its value as `false`.

Note: Specifying the parameter as `true` performs a cluster-wide operation. If an error occurs on any node, the function displays a warning message and skips the node where the error occurs. It then continues executing the operation on the remaining nodes.

Examples

This example shows how to set a storage policy for the table `test` to use the storage labeled SSD as its default location:

```sql
=> select set_object_storage_policy('test','ssd', true);
set_object_storage_policy
-----------------------------
Object storage policy set.
Task: moving storages
(Table: public.test) (Projection: public.test_b0)
(Table: public.test) (Projection: public.test_b1)
(1 row)
```
See Also

- `ALTER_LOCATION_LABEL`
- `CLEAR_OBJECT_STORAGE_POLICY`
- Creating Storage Policies
- `ENFORCE_OBJECT_STORAGE_POLICY`
- Moving Data Storage Locations

System Table Restriction Functions

This section contains the functions associated with the System Table Restriction and Access functionality.

In This Section

RELEASE_SYSTEM_TABLES_ACCESS

Opens access to non-superuser-only tables that are not accessible during lockdown.

Syntax

`RELEASE_SYSTEM_TABLES_ACCESS()`

Privileges

Superuser

Determine if a Table is Restricted

Query the `SYSTEM_TABLES` table to determine if a table is accessible. The table contains a column called `is_accessible_during_lockdown`. This flag indicates if a table is accessible. Valid values are:

- `t` — A table is accessible
f — A table is not accessible

Examples

The following examples show how to open access to system tables:

```sql
=> SELECT RELEASE_SYSTEM_TABLES_ACCESS();
RELEASE_SYSTEM_TABLES_ACCESS
---------------------------------------
Granted SELECT privileges on system tables to public.
(1 row)
```

See Also

- `RESTRICT_SYSTEM_TABLES_ACCESS`
- System Table Restriction and Access

**RESTRICT_SYSTEM_TABLES_ACCESS**

Restricts access to non-superuser-only tables that are not accessible during lockdown.

Syntax

```sql
RESTRICT_SYSTEM_TABLES_ACCESS()
```

Privileges

Superuser

Determine if a Table is Restricted

Query the `SYSTEM_TABLES` table to determine if a table is accessible. The table contains a column called `is_accessible_during_lockdown`. This flag indicates if a table is accessible. Valid values are:

- `t` — A table is accessible
- `f` — A table is not accessible

Examples

The following examples show how to restrict system table access:
See Also

- RELEASE_SYSTEM_TABLES_ACCESS
- System Table Restriction and Access

Text Search Functions

This section contains text search functions specific to Vertica.

DELETE_TOKENIZER_CONFIG_FILE

Deletes a tokenizer configuration file.

Syntax

```sql
SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE (USING PARAMETERS proc_oid='proc_oid', confirm={true | false});
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>confirm</td>
<td>Boolean flag. Indicates that the configuration file should be removed even if the tokenizer is still in use. True — Force deletion of the tokenizer when the used parameter value is True. False — Delete tokenizer if the used parameter value is False. Default: False</td>
</tr>
<tr>
<td>proc_oid</td>
<td>A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information.</td>
</tr>
</tbody>
</table>
Examples

The following example shows how you can use `DELETE_TOKENIZER_CONFIG_FILE` to delete the tokenizer configuration file:

```sql
=> SELECT v_txtindex.DELETE_TOKENIZER_CONFIG_FILE (USING PARAMETERS proc_oid='45035996274126984');
DELETE_TOKENIZER_CONFIG_FILE
---------
t
(1 row)
```

GET_TOKENIZER_PARAMETER

Returns the configuration parameter for a given tokenizer.

Syntax

```sql
SELECT v_txtindex.GET_TOKENIZER_PARAMETER(parameter_name USING PARAMETERS proc_oid='proc_oid');
```

Parameters

<table>
<thead>
<tr>
<th>parameter_name</th>
<th>Name of the parameter to be returned. One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopWordsCaseInsensitive</td>
<td></td>
</tr>
<tr>
<td>minorSeparators</td>
<td></td>
</tr>
<tr>
<td>majorSeparators</td>
<td></td>
</tr>
<tr>
<td>minLength</td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td></td>
</tr>
<tr>
<td>ngramsSize</td>
<td></td>
</tr>
<tr>
<td>used</td>
<td></td>
</tr>
</tbody>
</table>

| proc_oid | A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information. |
**Examples**

The following examples show how you can use `GET_TOKENIZER_PARAMETER`.

**Return the stop words used in a tokenizer:**

```sql
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('stopwordcaseinsensitive' USING PARAMETERS proc_oid='45035996274126984');
getTokenizerParameter
------------------------
devil,TODAY,the,fox
(1 row)
```

**Return the major separators used in a tokenizer:**

```sql
=> SELECT v_txtindex.GET_TOKENIZER_PARAMETER('majorseparators' USING PARAMETERS proc_oid='45035996274126984');
getTokenizerParameter
------------------------
{}[]&[]
(1 row)
```

**READ_CONFIG_FILE**

Reads and returns the key-value pairs of all the parameters of a given tokenizer.

You must use the OVER() clause with this function.

**Syntax**

```sql
SELECT v_txtindex.READ_CONFIG_FILE(USING PARAMETERS proc_oid='proc_oid') OVER ()
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proc_oid</code></td>
<td>A unique identifier assigned to a tokenizer when it is created. Users must query the system table <code>vs_procedures</code> to get the proc_oid for a given tokenizer name. See <a href="#">Configuring a Tokenizer</a> for more information.</td>
</tr>
</tbody>
</table>

**Examples**

The following example shows how you can use `READ_CONFIG_FILE` to return the parameters associated with a tokenizer:
SELECT v_txtindex.READ_CONFIG_FILE(USING PARAMETERS proc_oid='45035996274126984') OVER()

<table>
<thead>
<tr>
<th>config_key</th>
<th>config_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>majorseparators</td>
<td>{ }&amp;[ ]()</td>
</tr>
<tr>
<td>stopwordsCaseInsensitive</td>
<td>devil,TODAY,the,fox</td>
</tr>
</tbody>
</table>

(2 rows)

**SET_TOKENIZER_PARAMETER**

Configures the tokenizer parameters.

**Important:** \n, \t, \r  must be entered as Unicode using Vertica notation, U&’\00D’, or using Vertica escaping notation, E’\r’. Otherwise, they are taken literally as two separate characters. For example, "\" & "\r".

**Syntax**

SELECT v_txtindex.SET_TOKENIZER_PARAMETER (parameter_name, parameter_value USING PARAMETERS proc_oid='proc_oid')

**Parameters**

<table>
<thead>
<tr>
<th>parameter_name</th>
<th>Name of the parameter to be configured.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Use one of the following:</td>
</tr>
<tr>
<td>stopwordsCaseInsensitive</td>
<td>List of stop words. All the tokens</td>
</tr>
<tr>
<td></td>
<td>that belong to the list are ignored. Vertica supports separators and stop words up to the first 256 Unicode characters.</td>
</tr>
<tr>
<td></td>
<td>If you want to define a stop word that contains a comma or a backslash, then it needs to be escaped.</td>
</tr>
<tr>
<td></td>
<td>For example: &quot;Dear Jack,&quot; &quot;Dear Jack&quot;</td>
</tr>
<tr>
<td>Default:</td>
<td>' ' (empty list)</td>
</tr>
<tr>
<td>majorSeparators</td>
<td>List of major separators. Enclose in quotes with no spaces between.</td>
</tr>
<tr>
<td>Default:</td>
<td>E' [ ]&lt;&gt;(){}</td>
</tr>
<tr>
<td>minorSeparators</td>
<td>List of minor separators. Enclose in quotes with no spaces between.</td>
</tr>
</tbody>
</table>
Default: `E'/:=@.-#$%\_`

- **minLength** — Minimum length a token can have, type Integer. Must be greater than 0.
  
  Default: `'2'`

- **maxLength** — Maximum length a token can be. Type Integer. Cannot be greater than 1024 bytes. For information about increasing the token size, see Text Search Parameters.
  
  Default: `'128'`

- **ngramsSize** — Integer value greater than zero. Use only with ngram tokenizers.
  
  Default: `'3'`

- **used** — Indicates when a tokenizer configuration cannot be changed. Type Boolean. After you set used to True, any calls to setTokenizerParameter fail.

  You must set the parameter used to True before using the configured tokenizer. Doing so prevents the configuration from being modified after being used to create a text index.

  Default: `False`

<table>
<thead>
<tr>
<th>parameter_value</th>
<th>The value of a configuration parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you want to disable minorSeperators or stopWordsCaseInsensitive, then set their values to `''.</td>
</tr>
</tbody>
</table>

| proc_oid | A unique identifier assigned to a tokenizer when it is created. Users must query the system table vs_procedures to get the proc_oid for a given tokenizer name. See Configuring a Tokenizer for more information. |

**Examples**

The following examples show how you can use `SET_TOKENIZER_PARAMETER` to configure stop words and separators.

Configure the stop words of a tokenizer:
Configure the major separators of a tokenizer:

```
=> SELECT v_txtindex.SET_TOKENIZER_PARAMETER('stopwordsCaseInsensitive', 'devil,TODAY,the,fox' USING PARAMETERS proc_oid='45035996274126984');

```

```
SELECT v_txtindex.SET_TOKENIZER_PARAMETER('majorSeparators',E'{}()&[]' USING PARAMETERS proc_oid='45035996274126984');

```

Table Management Functions

This section contains the functions associated with the Vertica library table management.

**COPY_TABLE**

Copies one table to another. This lightweight, in-memory copy increases performance by initially sharing the same storage between two tables. The copied table includes copies of any explicitly created projections from the source table. Once copied, the source and copy tables are independent of each other. Users can perform operations on one table without impacting the other. These operations can increase the overall storage required for both tables.

Creating multiple, concurrent copies of the same table may cause some of the copies to fail. To ensure success, copy tables sequentially.

**Note:** Although they share storage space, Vertica considers the tables as discrete objects for license capacity purposes. For example, copying a one TB table would only consume one TB of space. Your Vertica license, however, considers them as separate objects consuming two TB of space.

**Syntax**

```
COPY_TABLE (  
    '[[database.]schema.]source-table',  
    '[[database.]schema.]target-table'  
)
```
Parameters

| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:
|                  | myschema.thisDbObject |
| source-table     | The source table to copy. Vertica copies all data from this table to the target table. |
| target-table     | The target table of the source table. If the target table already exists, Vertica appends the source to the existing table. If the table does not exist, Vertica creates a table from the source table's definition, by calling CREATE TABLE with LIKE and INCLUDING PROJECTIONS clause. The new table inherits ownership from the source table. For details, see Replicating a Table. |

Privileges

- User must have INSERT, UPDATE, DELETE and SELECT privileges on the source table.
- User must have CREATE privileges on the target table and target table schema if COPY_TABLE is creating a new table.
- User must have INSERT privileges on the target table if COPY_TABLE is adding to an existing table.

Table Attribute Requirements

The following attributes of both tables must be identical:

- Column definitions, including NULL/NOT NULL constraints
- Segmentation
- Partitioning expression
- Number of projections
- Projection sort order
- Primary and unique key constraints. However, the key constraints do not have to be identically enabled.

  Note: If the target table has primary or unique key constraints enabled and moving the partitions will insert duplicate key values into the target table, Vertica rolls back the operation. Enforcing constraints requires disk reads and can slow the copy process.

- Number and definitions of text indices.

Table Restrictions

The following restrictions apply to the source and target tables:

- If the source and target partitions are in different storage tiers, Vertica returns a warning but the operation proceeds. The partitions remain in their existing storage tier.

- If the source table contains a sequence, Vertica converts the sequence to an integer before copying it to the target table. If the target table contains auto-increment, identity, or named sequence columns, Vertica cancels the copy and displays an error message.

- The following tables cannot be used as sources or targets:
  - Temporary tables
  - Virtual tables
  - System tables
  - External tables

Examples

If you call COPY_TABLE and the target table does not exist, the function creates the table automatically. In the following example, the target table `public.newtable` does not exist. COPY_TABLE creates the table and replicates the source table. Vertica also copies all the constraints associated with the source table except foreign key constraints.

```sql
=> SELECT COPY_TABLE(
  'public.product_dimension',
  'public.newtable');
```
REBALANCE_TABLE

Synchronously rebalances data in the specified table.

A rebalance operation performs the following tasks:

- Distributes data based on:
  - User-defined fault groups, if specified
  - Large cluster automatic fault groups
- Redistributes database projection data across all nodes.

Syntax

REBALANCE_TABLE('[[database.]schema.]table-name')

Parameters

<table>
<thead>
<tr>
<th>schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

| table-name | The table to rebalance.                                                                                               |

Privileges

Superuser

When to Rebalance

Rebalancing is useful or even necessary after you perform the following tasks:
- Mark one or more nodes as ephemeral in preparation of removing them from the cluster.
- Add one or more nodes to the cluster so that Vertica can populate the empty nodes with data.
- Change the scaling factor of an elastic cluster, which determines the number of storage containers used to store a projection across the database.
- Set the control node size or realign control nodes on a large cluster layout
- Add nodes to or remove nodes from a fault group.

Tip: By default, before performing a rebalance, Vertica queries system tables to compute the size of all projections involved in the rebalance task. This query can add significant overhead to the rebalance operation. To disable this query, set projection configuration parameter RebalanceQueryStorageContainers to 0.

Example

The following command shows how to rebalance data on the specified table.

```sql
=> SELECT REBALANCE_TABLE('online_sales.online_sales_fact');
REBALANCE_TABLE
---------------------
REBALANCED
(1 row)
```

See Also

- REBALANCE_CLUSTER
- Rebalancing Data Across Nodes
- NODES
Tuple Mover Functions

This section contains tuple mover functions specific to Vertica.

DO_TM_TASK

Runs a Tuple Mover operation on the specified table or projection and commits any current transaction.

Tip: Running this function does not require you to stop the tuple mover.

Syntax

DO_TM_TASK('task', '[database.]schema.]table | projection')

Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>Specifies one of the following tuple mover operations:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- moveout: Moves out data from WOS to ROS. For details, see Moveout in the Administrator's Guide.</td>
</tr>
<tr>
<td></td>
<td>- mergeout: Consolidates ROS containers and purges deleted records. For details, see Mergeout in the Administrator's Guide.</td>
</tr>
<tr>
<td></td>
<td>- analyze_row_count: Collects the number of rows in the specified projection. If you specify a table name, DO_TM_TASK returns the row counts for all projections of that table. Vertica automatically analyzes all projection row counts at the time intervals specified by configuration parameter AnalyzeRowCountInterval.</td>
</tr>
<tr>
<td></td>
<td>DO_TM_TASK aggregates row counts calculated during loads. It commits this data to the catalog when the percentage of WOS to ROS equals the setting in configuration parameter ARCCommitPercentage.</td>
</tr>
</tbody>
</table>
analyze_row_count analyzes the row count of Vertica projections. To calculate row counts for external tables, use **ANALYZE_EXTERNAL_ROW_COUNT**.

[database.]schema

Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:

```sql
myschema.thisDbObject
```

If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>table</th>
<th>projection</th>
</tr>
</thead>
</table>

Applies task to the specified table or projection. If you specify a projection and it is not found, DO_TM_TASK looks for a table with that name and, if found, applies the task to it and all projections associated with it.

If no table or projection is specified, the task is applied to all tables and projections in the database.

Privileges

- Any INSERT/UPDATE/DELETE privilege on table
- USAGE privileges on schema

Examples

The following example performs a moveout of all projections for the t1 table:

```sql
=> SELECT DO_TM_TASK('moveout', 't1');
```

The following example performs a moveout for the t1_proj projection:

```sql
=> SELECT DO_TM_TASK('moveout', 't1_proj');
```
Workload Management Functions

This section contains workload management functions specific to Vertica.

**ANALYZE_WORKLOAD**

Runs the Workload Analyzer (WLA), a utility that analyzes system information held in system tables.

WLA intelligently monitors the performance of SQL queries and workload history, resources, and configurations to identify the root causes for poor query performance. ANALYZE_WORKLOAD returns tuning recommendations for all events within the scope and time that you specify, from system table TUNING_RECOMMENDATIONS.

Tuning recommendations are based on a combination of statistics, system and data collector events, and database-table-projection design. WLA recommendations can help you quickly and easily tune query performance.

See Understanding WLA Triggering Conditions in the Administrator's Guide for the common triggering conditions and recommendations.

**Syntax**

```
ANALYZE_WORKLOAD ( '[ scope ]' [, 'since-time' | save-data ] );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **scope** | Specifies the catalog objects to analyze, as follows:  
```
[[database.]schema.]table
```
If set to an empty string, Vertica returns recommendations for all database objects.
If you specify a database, it must be the current database. |
| **since-time** | Specifies the start time for the analysis time span, which continues up to the current system status, inclusive. If you omit this parameter, ANALYZE_WORKLOAD returns recommendations on events since the last time you called this function. |

**Note:** You must explicitly cast strings to TIMESTAMP or TIMESTAMPZ.
For example:

```
SELECT ANALYZE_WORKLOAD('T1', '2010-10-04 11:18:15':TIMESTAMPZ);
SELECT ANALYZE_WORKLOAD('T1', TIMESTAMP '2010-10-04 11:18:15');
```

### save-data

Specifies whether to save returned values from `ANALYZE_WORKLOAD`:

- **false** (default): Results are discarded.
- **true**: Saves the results returned by `ANALYZE_WORKLOAD`. Subsequent calls to `ANALYZE_WORKLOAD` return results that start from the last invocation when results were saved. Object events preceding that invocation are ignored.

### Return Values

Returns aggregated tuning recommendations from `TUNING_RECOMMENDATIONS`.

### Privileges

Superuser

### Examples

See [Getting Tuning Recommendations](#) in the Administrator's Guide.

### See Also

- Analyzing Workloads
- Understanding WLA Triggering Conditions

---

**CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY**

Changes the run-time priority of an active query.

**Note:** This function replaces deprecated function `CHANGE_RUNTIME_PRIORITY`.

### Syntax

```
CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY(transaction-id, 'value')
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>transaction-id</strong></td>
<td>Identifies the transaction, obtained from the system table <strong>SESSIONS</strong>.</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>The RUNTIMEPRIORITY value: HIGH, MEDIUM, or LOW.</td>
</tr>
</tbody>
</table>

Privileges

- Superuser: None
- Non-superusers can only change the runtime priority of their own queries, and cannot raise the runtime priority of a query to a level higher than that of the resource pool.

Example

See Changing Runtime Priority of a Running Query.

**CHANGE_RUNTIME_PRIORITY**

Changes the run-time priority of a query that is actively running. Note that, while this function is still valid, you should instead use CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY to change run-time priority. CHANGE_RUNTIME_PRIORITY will be deprecated in a future release of Vertica.

Syntax

```
CHANGE_RUNTIME_PRIORITY(TRANSACTION_ID, STATEMENT_ID, 'value')
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_ID</td>
<td>An identifier for the transaction within the session.</td>
</tr>
<tr>
<td></td>
<td>TRANSACTION_ID cannot be NULL.</td>
</tr>
<tr>
<td></td>
<td>You can find the transaction ID in the Sessions table.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the currently executing statement.</td>
</tr>
</tbody>
</table>
You can find the statement ID in the Sessions table.
You can specify NULL to change the run-time priority of the currently running query within the transaction.

| 'value' | The RUNTIMEPRIORITY value. Can be HIGH, MEDIUM, or LOW. |

privileges

No special privileges required. However, non-superusers can change the run-time priority of their own queries only. In addition, non-superusers can never raise the run-time priority of a query to a level higher than that of the resource pool.

Example

```sql
==> SELECT CHANGE_RUNTIME_PRIORITY(45035996273705748, NULL, 'low');
```

**MOVE_STATEMENT_TO_RESOURCE_POOL**

Attempts to move the specified query to the specified target pool.

Syntax

```
MOVE_STATEMENT_TO_RESOURCE_POOL (session_id, transaction_id, statement_id, target_resource_pool_name)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>Identifier for the session where the query you want to move is currently executing.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>Identifier for the transaction within the session.</td>
</tr>
<tr>
<td>statement_id</td>
<td>Unique numeric ID for the statement you want to move.</td>
</tr>
<tr>
<td>target_resource_pool_name</td>
<td>Name of the existing resource pool to which you want to move the specified query.</td>
</tr>
</tbody>
</table>

Outputs

The function may return the following results:
MOV_REPLAN: Target pool does not have sufficient resources. See v_monitor.resource_pool_move for details. Vertica will attempt to replan the statement on target pool.

MOV_REPLAN: Target pool has priority HOLD. Vertica will attempt to replan the statement on target pool.

MOV_FAILED: Statement not found.

MOV_NO_OP: Statement already on target pool.

MOV_REPLAN: Statement is in queue. Vertica will attempt to replan the statement on target pool.

MOV_SUCCEED: Statement successfully moved to target pool.

Privileges
Superuser

Examples

The following example shows how you can move a specific statement to a resource pool called my_target_pool:

```sql
=> SELECT MOVE_STATEMENT_TO_RESOURCE_POOL ('v_vmart_node0001.example.-31427:0x82fbm', 45035996273711993, 1, 'my_target_pool');
```

See Also:
- Manually Moving Queries to Different Resource Pools
- RESOURCE_POOL_MOVE

SLEEP

Waits a specified number of seconds before executing another statement or command.

Syntax

```sql
SLEEP( seconds )
```
Parameters

| seconds | The wait time, specified in one or more seconds (0 or higher) expressed as a positive integer. Single quotes are optional; for example, SLEEP(3) is the same as SLEEP(’3’). |

Notes

- This function returns value 0 when successful; otherwise it returns an error message due to syntax errors.
- You cannot cancel a sleep operation.
- Be cautious when using SLEEP() in an environment with shared resources, such as in combination with transactions that take exclusive locks.

Example

The following command suspends execution for 100 seconds:

```sql
=> SELECT SLEEP(100);
sleep
-----
 0
(1 row)
```
SQL Statements

The primary structure of a SQL query is its statement. Multiple statements are separated by semicolons. The following example contains four common SQL statements—CREATE TABLE, INSERT, SELECT, and COMMIT:

```sql
=> CREATE TABLE comments (id INT, comment VARCHAR);
CREATE TABLE
=> INSERT INTO comments VALUES (1, 'Hello World');
OUTPUT
-------
1
(1 row)
=> SELECT * FROM comments;
id | comment
------------
1 | Hello World
(1 row)
=> COMMIT;
COMMIT
=>
```

ALTER ACCESS POLICY

The ALTER ACCESS POLICY statement:

- Enables and disables individual access policies in a table.
- Copies an access policy from one table to another.

Important: When you copy or rename a table, access policies associated with the original table are not included in the copied table. You must use ALTER ACCESS POLICY to copy the access policies to the new table.

Syntax

```
ALTER ACCESS POLICY ON [schema.]tablename
    | FOR COLUMN { columnname expression }
    | FOR ROWS WHERE expression
    [ENABLE | DISABLE | COPY TO TABLE tablename];
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>The name of the table that contains the access policy you want to enable, disable, or copy.</td>
</tr>
<tr>
<td>columnname</td>
<td>Either of the following values:</td>
</tr>
<tr>
<td></td>
<td>• The name of the column on which to enable or disable the access policy</td>
</tr>
<tr>
<td></td>
<td>• The name of the column you want to copy</td>
</tr>
<tr>
<td>expression</td>
<td>An expression that provides further information to limit column or row access.</td>
</tr>
<tr>
<td></td>
<td>• In a column access policy the expression is the transformation of the column.</td>
</tr>
<tr>
<td></td>
<td>• In a row access policy the expressions is the content of the WHERE clause.</td>
</tr>
<tr>
<td></td>
<td>For example, the expression:</td>
</tr>
<tr>
<td></td>
<td>=&gt; ALTER ACCESS POLICY ON customer_dimension FOR COLUMN customer_key length ('xxxxx') enable;</td>
</tr>
<tr>
<td></td>
<td>limits access to strings in the customer_key column to a specific length.</td>
</tr>
<tr>
<td>ENABLE</td>
<td>DISABLE</td>
</tr>
<tr>
<td>COPY TO TABLE tablename</td>
<td>Copies the existing access policy to the specified table. Do not use an expression when performing a COPY TO function.</td>
</tr>
</tbody>
</table>

Privileges

You must be a dbadmin user to alter an access policy.
Examples

Copy Access Policy from Table to Table

=> ALTER ACCESS POLICY ON customer FOR COLUMN customer_number COPY TO TABLE customer_old;

Enable Access Policy

=> ALTER ACCESS POLICY ON customer FOR ROWS WHERE cid1>1 ENABLE;

ALTER AUTHENTICATION

Modifies the settings for a specified authentication method.

Syntax

ALTER AUTHENTICATION auth_method_name {
    |   { ENABLE | DISABLE }
    |   { LOCAL | HOST [ { TLS | NO TLS } ] host_ip_address }
    |   RENAME TO new_auth_method_name
    |   METHOD value
    |   SET param=value[, ...]
    |   PRIORITY value
}

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_method_name</td>
<td>Name of the authentication method that you want to create.</td>
</tr>
<tr>
<td></td>
<td><strong>Type:</strong> VARCHAR</td>
</tr>
<tr>
<td>ENABLE</td>
<td>DISABLE</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> Enabled</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> When you perform an upgrade and use Kerberos authentication, you must manually set the authentication to ENABLE as it is disabled by default.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>LOCAL</td>
<td>HOST [ { TLS</td>
</tr>
<tr>
<td>RENAME TO new_auth_method_name</td>
<td>Rename the authentication record. <strong>Type</strong>: VARCHAR</td>
</tr>
<tr>
<td>METHOD value</td>
<td>The authentication method you are altering.</td>
</tr>
<tr>
<td>SET param=value</td>
<td>Set a parameter name and value for the authentication method that you are creating. Required only for LDAP and Ident authentication. ALTER AUTHENTICATION validates the parameters you enter. See parameters for specific authentication types in Client Authentication.</td>
</tr>
<tr>
<td>PRIORITY value</td>
<td>If the user is associated with multiple authentication methods, the priority value specifies which authentication method Vertica tries first. <strong>Default</strong>: 0 <strong>Type</strong>: INTEGER Higher values indicate higher priorities. For example, a priority of 10 is higher than a priority of 5; priority 0 is the lowest possible value. For details, see Priorities for Client Authentication Methods.</td>
</tr>
</tbody>
</table>

**Privileges**

Must have DBADMIN privileges.
Examples

Enabling and Disabling Authentication Methods

This example uses ALTER AUTHENTICATION to disable the v_ldap authentication method and then enable it again:

```
=> ALTER AUTHENTICATION v_ldap DISABLE;
=> ALTER AUTHENTICATION v_ldap ENABLE;
```

Renaming Authentication Methods

This example renames the v_kerberos authentication method to K5 and enables it. All users who have been granted the v_kerberos authentication method now have the K5 method granted instead.

```
=> ALTER AUTHENTICATION v_kerberos RENAME TO K5 ENABLE;
```

Modifying Authentication Parameters

This example sets the system user for ident1 authentication to user1:

```
=> CREATE AUTHENTICATION ident1 METHOD 'ident' LOCAL;
=> ALTER AUTHENTICATION ident1 SET system_users='user1';
```

When you set or modify LDAP or Ident parameters using ALTER AUTHENTICATION, Vertica validates them.

This example changes the IP address and specifies the parameters for an LDAP authentication method named Ldap1. Specify the bind parameters for the LDAP server. Vertica connects to the LDAP server, which authenticates the database client. If authentication succeeds, Vertica authenticates any users who have been associated with (granted) the Ldap1 authentication method on the designated LDAP server:

```
=> CREATE AUTHENTICATION Ldap1 METHOD 'ldap' HOST '172.16.65.196';
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',
   binddn_prefix='cn=', binddn_suffix=',dc=qa_domain,dc=com';
```

The next example specifies the parameters for an LDAP authentication method named Ldap2. Specify the LDAP search and bind parameters. Sometimes, Vertica does not have enough information to create the distinguished name (DN) for a user attempting to authenticate. In such cases, you must specify to use LDAP search and bind:

```
=> CREATE AUTHENTICATION Ldap2 METHOD 'ldap' HOST '172.16.65.196';
```
=> ALTER AUTHENTICATION Ldap2 SET basedn='dc=qa_domain,dc=com', binddn='cn=Manager,dc=qa_domain, dc=com', search_attribute='cn', bind_password='secret';

### Changing the Authentication Method

This example changes the `localpwd` authentication from hash to trust:

```sql
=> CREATE AUTHENTICATION localpwd METHOD 'hash' LOCAL;
=> ALTER AUTHENTICATION localpwd METHOD 'trust';
```

### Set Multiple Realms

This example sets another realm for the authentication method `krb_local`:

```sql
=> ALTER AUTHENTICATION krb_local set realm = 'COMPANY.COM';
```

### See Also

- `CREATE AUTHENTICATION`
- `DROP AUTHENTICATION`
- `GRANT (Authentication)`
- `REVOKE (Authentication)`

### ALTER DATABASE

Use `ALTER DATABASE` to perform the following tasks:

- Set and clear database configuration parameters.
- Drop all fault groups and their child fault groups from a database.
- Specify the subnet name of a public network to use for import/export.
- Restore down nodes, and revert active standby nodes to standby status.
Syntax

```
ALTER DATABASE db-spec {
    DROP ALL FAULT GROUP
    | EXPORT ON { subnet-name | DEFAULT }
    | RESET STANDBY
    | SET [PARAMETER] parameter=value [,...]
    | CLEAR [PARAMETER] parameter[,...]
}
```

Parameters

<table>
<thead>
<tr>
<th>db-spec</th>
<th>Specifies the database to alter, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The database name</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT: The current database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DROP ALL FAULT GROUP</th>
<th>Drops all fault groups defined on the specified database. The syntax for DROP ALL FAULT GROUP is singular for GROUP.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EXPORT ON</th>
<th>Specifies the network to use for importing and exporting data, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• subnet-name: A subnet of the public network.</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT: Specifies to use a private network.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESET STANDBY</th>
<th>Restores any replaced nodes and reverts all active standby nodes to standby status. If any replaced nodes cannot resume activity, Vertica leaves the standby nodes in place.</th>
</tr>
</thead>
</table>

<p>| SET [PARAMETER]      | Sets one or more configuration parameters to the specified value at the database level.                        |</p>
<table>
<thead>
<tr>
<th>parameter=value</th>
<th></th>
</tr>
</thead>
</table>

| CLEAR [PARAMETER]    | Clears one or more specified configuration parameters at the database level.                                   |
| parameter             |                                                                                                               |

Privileges

Superuser
Examples

Drop and restore all default groups:

```sql
=> ALTER DATABASE exampledb DROP ALL FAULT GROUP;
ALTER DATABASE
```

Restore down nodes and revert all active standby nodes:

```sql
=> ALTER DATABASE exampledb RESET STANDBY;
ALTER DATABASE
```

Set multiple configuration parameters:

```sql
=> ALTER DATABASE exampledb SET
   AnalyzeRowCountInterval = 3600,
   DBDCorrelationSampleRowCount = 3000,
   ActivePartitionCount = 2;
```

Clear configuration parameters:

```sql
=> ALTER DATABASE exampledb CLEAR
   AnalyzeRowCountInterval, DBDCorrelationSampleRowCount, ActivePartitionCount;
```

**ALTER FAULT GROUP**

Modifies an existing fault group. For example, use the ALTER FAULT GROUP statement to:

- Add a node to or drop a node from an existing fault group
- Add a child fault group to or drop a child fault group from a parent fault group
- Rename a fault group

**Syntax**

```
ALTER FAULT GROUP fault-group-name
   [ ADD NODE node-name ]
   [ DROP NODE node-name ]
   [ ADD FAULT GROUP child-fault-group-name ]
   [ DROP FAULT GROUP child-fault-group-name ]
   [ RENAME TO new-fault-group-name ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault-group-name</td>
<td>The existing fault group name you want to modify.</td>
</tr>
<tr>
<td></td>
<td>Tip: For a list of all fault groups defined in the cluster, query the V_CATALOG.FAULT_GROUPS system table.</td>
</tr>
<tr>
<td>node-name</td>
<td>The node name you want to add to or drop from the existing (parent) fault group.</td>
</tr>
<tr>
<td>child-fault-group-name</td>
<td>The name of the child fault group you want to add to or remove from an existing parent fault group.</td>
</tr>
<tr>
<td>new-fault-group-name</td>
<td>The new name for the fault group you want to rename.</td>
</tr>
</tbody>
</table>

Privileges

Must be a superuser to alter a fault group.

Example

This example renames the parent0 fault group to parent100:

```
=> ALTER FAULT GROUP parent0 RENAME TO parent100;
ALTER FAULT GROUP
```

You can verify the change by querying the V_CATALOG.FAULT_GROUPS system table:

```
=> SELECT member_name FROM fault_groups;
   member_name
------------------------
   v_exampledb_node0003
   parent100
   mygroup
(3 rows)
```
See Also

- CREATE FAULT GROUP
- V_CATALOG.FAULT_GROUPS
- V_CATALOG.CLUSTER_LAYOUT
- Fault Groups
- High Availability With Fault Groups

ALTER FUNCTION (SQL)

Alters a user-defined SQL function.

Syntax

ALTER FUNCTION [[db-name.]schema.]function-name([arg-type1], ...) {SET SCHEMA new_schema
  | RENAME TO new_name
  | OWNER TO new_owner}

Parameters

<table>
<thead>
<tr>
<th>[db-name.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>function-name</td>
<td>The name of the SQL function to alter. Each function requires an accompanying argument, so you must specify the argument type for each function.</td>
</tr>
<tr>
<td>RENAME TO new_name</td>
<td>Specifies the new name of the function</td>
</tr>
<tr>
<td>SET SCHEMA new_schema</td>
<td>Specifies the new schema name where the function resides.</td>
</tr>
</tbody>
</table>
OWNER TO new_owner

Specifies the new owner of the function.

Privileges

- Only a superuser or owner can alter a function.

- To rename a function (ALTER FUNCTION RENAME TO) the user must have USAGE and CREATE privilege on schema that contains the function.

- To specify a new schema (ALTER FUNCTION SET SCHEMA), the user must have USAGE privilege on schema that currently contains the function (old schema) and CREATE privilege on the schema to which the function will be moved (new schema).

Examples

This example renames a function called SQL_one to SQL_two:

```sql
=> ALTER FUNCTION SQL_one (int,int) RENAME TO SQL_two;
```

This example moves the SQL_two function to a new schema called macros:

```sql
=> ALTER FUNCTION SQL_two (int) SET SCHEMA macros;
```

This example assigns a new owner to SQL_two:

```sql
=> ALTER FUNCTION SQL_two (int, int) OWNER TO user1;
```

See Also

- CREATE FUNCTION (SQL Functions)
- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
**USER_FUNCTIONS**

**Using User-Defined SQL Functions**

**ALTER FUNCTION (UDF)**

Alters a user defined function (UDF).

**Syntax**

```
ALTER FUNCTION [[db-name.]schema.]function-name([arg-type1,] ...) 
  { SET SCHEMA new_schema 
  | RENAME TO new_name 
  | SET FENCED bool_val 
  | OWNER TO new_owner }
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[db-name.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>function-name</code></td>
<td>The name of the UDF to alter. You must specify the argument type list, which may be empty.</td>
</tr>
<tr>
<td>RENAME TO <code>new_name</code></td>
<td>Specifies the new name of the function</td>
</tr>
<tr>
<td>SET SCHEMA <code>new_schema</code></td>
<td>Specifies the new schema name where the function resides.</td>
</tr>
<tr>
<td>OWNER TO <code>new_owner</code></td>
<td>Specifies the new owner of the function.</td>
</tr>
</tbody>
</table>
| SET FENCED `bool_val`      | A boolean value that specifies if Fenced Mode is enabled for this function. Valid values are: true - enables Fenced Mode  
false - disables Fenced Mode  
Fenced Mode is not available for User Defined Aggregates or User
Privileges

- Only a superuser or owner can alter a function.
- To rename a function (ALTER FUNCTION RENAME TO) the user must have USAGE and CREATE privilege on schema that contains the function.
- To specify a new schema (ALTER FUNCTION SET SCHEMA), the user must have USAGE privilege on the schema that currently contains the function (old schema) and CREATE privilege on the schema to which the function will be moved (new schema).

Examples

This example renames a function called UDF_one to UDF_two:

```sql
=> ALTER FUNCTION UDF_one (int,int) RENAME TO UDF_two;
```

This example moves the UDF_two function to a new schema called macros:

```sql
=> ALTER FUNCTION UDF_two (int) SET SCHEMA macros;
```

This example disables Fenced Mode for the UDF_two function:

```sql
=> ALTER FUNCTION UDF_two (int, int) SET FENCED false;
```

See Also

- CREATE FUNCTION (UDF)
- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
ALTER HCATALOG SCHEMA

Alters parameter values on a schema that was created with CREATE HCATALOG SCHEMA. HCatalog schemas are used by the HCatalog Connector to access data stored in a Hive data warehouse. For more information, see Using the HCatalog Connector in Integrating with Apache Hadoop.

Some parameters cannot be altered after creation. If you need to change one of those values, delete and recreate the schema instead. You can use ALTER HCATALOG SCHEMA to change the following parameters:

- HOSTNAME
- PORT
- HIVESERVER2_HOSTNAME
- WEBSERVICE_HOSTNAME
- WEBSERVICE_PORT
- WEBHDFS_ADDRESS
- HCATALOG_CONNECTION_TIMEOUT
- HCATALOG_SLOW_TRANSFER_LIMIT
- HCATALOG_SLOW_TRANSFER_TIME
- SSL_CONFIG

Syntax

ALTER HCATALOG SCHEMA schema-name SET [param=value]+;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x)
### ALTER HCATALOG SCHEMA

The name of the schema in the Vertica catalog to alter. The tables in the Hive database are available through this schema.

<table>
<thead>
<tr>
<th>param</th>
<th>The name of the parameter to alter.</th>
</tr>
</thead>
</table>
| value       | The new value for the parameter. You must specify a value; this statement does not read default values from configuration files like CREATE HCATALOG SCHEMA.

### Privileges

One of the following:

- Superuser
- Schema owner

### Examples

The following example shows how to change the Hive metastore hostname and port for the "hcat" schema. In this example, Hive uses High Availability metastore.

```sql
=> ALTER HCATALOG SCHEMA hcat SET HOSTNAME='thrift://ms1.example.com:9083,thrift://ms2.example.com:9083';
```

The following example shows the error you receive if you try to set an unalterable parameter.

```sql
=> ALTER HCATALOG SCHEMA hcat SET HCATALOG_USER='admin';
ERROR 4856: Syntax error at or near "HCATALOG_USER" at character 39
```

### ALTER LIBRARY

Replaces the library file (C++ .so file, Java JAR file, or R source file) associated with a UDx library that is defined in the Vertica catalog. Vertica automatically distributes copies of the updated file throughout the cluster. UDxs defined in the catalog that reference the updated library automatically start using the updated library file. Nodes that are down or added to the cluster also receive a copy of the updated library file as soon as they join the cluster.
Syntax

ALTER LIBRARY [[database.]schema.]library-name [DEPENDS 'support-path'] AS 'library-path';

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>library-name</td>
<td>The library to alter. You can only alter an existing library.</td>
</tr>
</tbody>
</table>
| DEPENDS 'support-path' | Indicates that the UDx library depends on one or more support libraries, where support-path specifies one or more absolute paths to the support libraries files, located in the initiator node's file system. You can specify multiple support paths as follows:  
  - Separate multiple paths with colons (:).  
  - Specify a directory that contains multiple libraries with an asterisk wildcard (*). For example: /home/mydir/mylibs/* |
| library-path   | The absolute path in the initiator node file system to the replacement library file.                                                            |
Requirements

- The new library must be developed in the same programming language as the library file being replaced. For example, you cannot use this statement to replace a C++ library file with an R library file.

- Vertica does not compare the functions defined in the new library to ensure they match any currently-defined functions in the catalog. If you change the signature of a function in the library (for example, if you change the number and data types accepted by a UDSF defined in the library), calls to that function will likely generate errors. If your new library file changes the definition of a function, you must remove the function using DROP FUNCTION before using ALTER LIBRARY to load the new library. You can then recreate the function using its new signature.

Examples

This example shows how to update an already-defined library myFunctions with a new file.

```sql
=> ALTER LIBRARY myFunctions AS '/home/dbadmin/my_new_functions.so';
```

See Also

Developing User-Defined Extensions (UDxs)

**ALTER MODEL**

Allows users to rename an existing model, change owner parameters, and set schema to the model.

Syntax

```
ALTER MODEL [[db-name.]schema.]model-name
  [ OWNER TO owner-name
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$db-name.schema$</td>
<td>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>$myschema$</td>
</tr>
<tr>
<td>$model-name$</td>
<td>The model to alter.</td>
</tr>
<tr>
<td>$OWNER TO $owner-name$</td>
<td>Changes the model owner.</td>
</tr>
<tr>
<td>$RENAME TO$</td>
<td>Renames the model.</td>
</tr>
<tr>
<td>$SET SCHEMA$</td>
<td>Moves the model from one schema to another.</td>
</tr>
</tbody>
</table>

Privileges

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

Examples

This example shows how you can alter an existing model to rename the model.

```sql
=> ALTER MODEL mymodel RENAME to mykmeansmodel;
ALTER MODEL
```

This example shows how you can alter an existing model to change the owner.

```sql
=> ALTER MODEL mykmeansmodel OWNER TO user1;
ALTER MODEL
```

This example shows how you can alter an existing model to rename the model.

```sql
=> ALTER MODEL mykmeansmodel SET SCHEMA public;
ALTER MODEL
```
See Also

- Altering Models

**ALTER NETWORK INTERFACE**

Lets you rename a network interface.

**Syntax**

```
ALTER NETWORK INTERFACE network-interface-name RENAME TO new-network-interface-name
```

**Parameters**

<table>
<thead>
<tr>
<th>network-interface-name</th>
<th>The name of the existing network interface.</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-network-interface-name</td>
<td>The new name for the network interface.</td>
</tr>
</tbody>
</table>

**Privileges**

Must be a superuser to alter a network interface.

**Examples**

This example shows how to rename a network interface.

```
=> ALTER NETWORK INTERFACE myNetwork RENAME TO myNewNetwork;
```
ALTER NODE

Sets and clears node-level configuration parameters on the specified node. ALTER NODE also performs the following management tasks:

- Changes the node type.
- Specifies the network interface of the public network on individual nodes that are used for import and export.
- Replaces a down node.

Syntax

ALTER NODE node-name {
  EXPORT ON { network-interface | DEFAULT }
  [ IS ] node-type
  REPLACE [ WITH standby-node ]
  RESET
  SET [PARAMETER] parameter=value[,...]
  CLEAR [PARAMETER] parameter[,...]
}

Parameters

<table>
<thead>
<tr>
<th>node-name</th>
<th>The name of the node to alter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[IS] node-type</td>
<td>Changes the node type, where node-type is one of the following:</td>
</tr>
<tr>
<td></td>
<td>- PERMANENT (default): A node that is used to store data.</td>
</tr>
<tr>
<td></td>
<td>- EPHEMERAL: A node that is in transition from one type to another—typically, from PERMANENT to either STANDBY or EXECUTE.</td>
</tr>
<tr>
<td></td>
<td>- STANDBY: A node that is reserved to replace any node when it goes down. When used as a replacement node, Vertica changes its type to PERMANENT. A standby node stores no segments or data until it is called to replace a down node. At</td>
</tr>
</tbody>
</table>
that time, Vertica changes its type to PERMANENT. For more information, see *Active Standby Nodes*.

- **EXECUTE**: A node that is reserved for computation purposes only. An execute node contains no segments or data.

<table>
<thead>
<tr>
<th>EXPORT ON</th>
<th>Specifies the network to use for importing and exporting data, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- <em>network-interface</em>: The name of a network interface of the public network.</td>
</tr>
<tr>
<td></td>
<td>- DEFAULT: Use the default network interface of the public network, as specified by <em>ALTER DATABASE</em>.</td>
</tr>
</tbody>
</table>

| REPLACE [WITH *standby-node*] | Replaces the specified node with an available active standby node. If you omit the WITH clause, Vertica tries to find a replacement node from the same fault group as the down node. If you specify a node that is not down, Vertica ignores this statement. |

| RESET | Restores the specified down node and returns its replacement to standby status. If the down node is unable to resume activity, Vertica ignores this statement and leaves the standby node in place. |

| SET [PARAMETER] *parameter=value* | Sets one or more configuration parameters to the specified value at the node level. |

| CLEAR [PARAMETER] *parameter* | Clears one or more specified configuration parameters. |

**Privileges**

Superuser
Examples

Specify to use the default network interface of public network on v_vmart_node0001 for import/export operations:

```sql
=> ALTER NODE v_vmart_node0001 EXPORT ON DEFAULT;
```

Replace down node v_vmart_node0001 with an active standby node, then restore it:

```sql
=> ALTER NODE v_vmart_node0001 REPLACE WITH standby1;
...  
=> ALTER NODE v_vmart_node0001 RESET;
```

Set and clear configuration parameter `MaxClientSessions`:

```sql
=> ALTER NODE v_vmart_node0001 SET MaxClientSessions = 0;
...  
=> ALTER NODE v_vmart_node0001 CLEAR MaxClientSessions;
```

Set the node type as `EPHEMERAL`:

```sql
=> ALTER NODE v_vmart_node0001 IS EPHEMERAL;
```

**ALTER NOTIFIER**

Updates an existing notifier.

**Note:** To change the action URL associated with an existing identifier, drop the notifier and re-create it.

**Syntax**

```
ALTER NOTIFIER notifier-name parameter[...]
```

*parameter*

- [NO] CHECK COMMITTED
- ENABLE | DISABLE
- IDENTIFIED BY uuid
- MAXMEMORYSIZE `max-memory-size`
- MAXPAYLOAD `max-payload-size`
- PARAMETERS `adapter-params`
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifier-name</td>
<td>Specifies the notifier to update.</td>
</tr>
<tr>
<td>[NO] CHECK COMMITTED</td>
<td>Specifies to wait for delivery confirmation before sending the next message in the queue. Not all messaging systems support delivery confirmation.</td>
</tr>
<tr>
<td>ENABLE</td>
<td>DISABLE</td>
</tr>
<tr>
<td>IDENTIFIED BY uuid</td>
<td>Specifies the notifier's unique identifier. If set, all the messages published by this notifier have this attribute.</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>The maximum size of the internal notifier, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows:</td>
</tr>
<tr>
<td></td>
<td>[\text{MAXMEMORYSIZE} \text{integer}{K</td>
</tr>
<tr>
<td></td>
<td>If the queue exceeds this size, the notifier drops excess messages.</td>
</tr>
<tr>
<td>MAXPAYLOAD</td>
<td>The maximum size of the message, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows:</td>
</tr>
<tr>
<td></td>
<td>[\text{MAXPAYLOAD} \text{integer}{K</td>
</tr>
<tr>
<td></td>
<td>The default setting is adapter-specific—for example, 1 M for Kafka.</td>
</tr>
<tr>
<td>PARAMETERS 'adapter-params'</td>
<td>Specifies one or more optional adapter parameters that are passed as a string to the adapter. Adapter parameters apply only to the adapter associated with the notifier.</td>
</tr>
<tr>
<td></td>
<td>For Kafka notifiers, refer to Kafka and Vertica Configuration Settings.</td>
</tr>
</tbody>
</table>

Privileges

Database Administrator
Examples

Update the settings on an existing notifier:

```sql
=> ALTER NOTIFIER my_dc_notifier
   ENABLE
   MAXMEMORYSIZE '2G'
   IDENTIFIED BY 'f8b0278a-3282-4e1a-9c86-e0f3f042a971'
   CHECK COMMITTED;
```

See Also

- CREATE NOTIFIER
- DROP NOTIFIER

**ALTER PROJECTION RENAME**

Initiates a rename operation on the specified projection.

**Syntax**

```sql
ALTER PROJECTION [[database.]schema.]projection RENAME TO new-name
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>schema</strong></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>
### projection

The projection to change. See Projection Naming for projection name conventions.

### new-name

The new projection name.

---

**Privileges**

Superuser, or the following:

- Anchor table owner
- CREATE privilege on the schema

---

**Examples**

This example shows how to rename a projection.

```sql
=> CREATE TABLE tbl (x integer, y integer); CREATE TABLE

=> CREATE PROJECTION tbl_p AS SELECT x from tbl; CREATE PROJECTION

=> ALTER PROJECTION tbl_p RENAME to new_tbl_p; ALTER PROJECTION
```

---

**See Also**

CREATE PROJECTION

---

**ALTER PROFILE**

Changes a profile. All parameters that are not set in a profile inherit their setting from the default profile. You can use ALTER PROFILE to change the default profile.

---

**Syntax**

```sql
CREATE PROFILE name LIMIT [ password-parameter setting ]...
```
**password-parameter**

PASSWORD_LIFE_TIME
PASSWORD_GRACE_TIME
FAILED_LOGIN_ATTEMPTS
PASSWORD_LOCK_TIME
PASSWORD_REUSE_MAX
PASSWORD_REUSE_TIME
PASSWORD_MAX_LENGTH
PASSWORD_MIN_LENGTH
PASSWORD_MIN_UPPERCASE_LETTERS
PASSWORD_MIN_LOWERCASE_LETTERS
PASSWORD_MIN_DIGITS
PASSWORD_MIN_SYMBOLS

## Parameters

**Note:** To reset a parameter to inherit from the default profile, set its value to `default`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>The name of the profile to create, where <code>name</code> conforms to conventions described in Identifiers. To modify the default profile, set <code>name</code> to <code>default</code>. For example:</td>
</tr>
<tr>
<td></td>
<td><code>ALTER PROFILE DEFAULT LIMIT PASSWORD_MIN_SYMBOLS 1;</code></td>
</tr>
<tr>
<td><strong>PASSWORD_LIFE_TIME</strong></td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ≥ 1: The number of days a password remains valid.</td>
</tr>
<tr>
<td></td>
<td>- UNLIMITED: Password remains valid indefinitely.</td>
</tr>
<tr>
<td></td>
<td>After your password's lifetime and grace period expire, you must change your password on your next login, if you have not done so already.</td>
</tr>
<tr>
<td><strong>PASSWORD_GRACE_TIME</strong></td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ≥ 1: The number of days a password can be used after it expires.</td>
</tr>
<tr>
<td></td>
<td>- UNLIMITED: No grace period.</td>
</tr>
<tr>
<td><strong>FAILED_LOGIN_ATTEMPTS</strong></td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ≥ 1: The number of days a password remains valid.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>● ≥ 1: The number of consecutive failed login attempts Vertica allows before locking your account.</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: Vertica allows an unlimited number of failed login attempts.</td>
</tr>
<tr>
<td>PASSWORD_LOCK_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>● ≥ 1: The number of days your account is locked after too many failed login attempts. The account is automatically unlocked when the lock time elapses.</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: Account remains indefinitely inaccessible until a superuser manually unlocks it.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_MAX</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>● ≥ 1: The number of times you must change your password before you can reuse an earlier password.</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: You can reuse an earlier password without any intervening changes.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>● ≥ 1: The number of days that must pass after a password is set before you can reuse it.</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: You can reuse an earlier password immediately.</td>
</tr>
<tr>
<td>PASSWORD_MAX_LENGTH</td>
<td>The maximum number of characters allowed in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>● Integer between 8 and 100, inclusive</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: Maximum of 100 characters</td>
</tr>
<tr>
<td>PASSWORD_MIN_LENGTH</td>
<td>The minimum number of characters required in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>● 0 to PASSWORD_MAX_LENGTH</td>
</tr>
<tr>
<td></td>
<td>● UNLIMITED: Minimum of PASSWORD_MAX_LENGTH</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PASSWORD_MIN_LETTERS</td>
<td>Minimum number of letters (a-z and A-Z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_UPPERCASE_LETTERS</td>
<td>Minimum number of uppercase letters (A-Z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_LOWERCASE_LETTERS</td>
<td>Minimum number of lowercase letters (a-z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_DIGITS</td>
<td>Minimum number of digits (0-9) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_SYMBOLS</td>
<td>Minimum number of symbols— printable non-letter and non-digit characters such as $, #, @— that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser
Profile Settings and Client Authentication

The following profile settings affect client authentication methods, such as LDAP or GSS:

- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME

All other profile settings are used only by Vertica to manage its passwords.

Example

```sql
ALTER PROFILE sample_profile LIMIT FAILED_LOGIN_ATTEMPTS 3;
```

See Also

- CREATE PROFILE
- DROP PROFILE
- Creating a Database Name and Password

**ALTER PROFILE RENAME**

Rename an existing profile.

**Syntax**

```
ALTER PROFILE name RENAME TO newname;
```

**Parameters**

| name     | The current name of the profile. |
The new name for the profile.

Privileges

Must be a superuser to alter a profile.

Examples

This example shows how to rename an existing profile.

```
ALTER PROFILE sample_profile RENAME TO new_sample_profile;
```

See Also

- ALTER PROFILE
- CREATE PROFILE
- DROP PROFILE

**ALTER RESOURCE POOL**

Modifies an existing resource pool by setting one or more parameters.

Syntax

```
ALTER RESOURCE POOL pool-name [ parameter-name setting ]...
```

Parameters

Note: You can set all resource pool parameters to their DEFAULT value. The `V_CATALOG RESOURCE_POOL_DEFAULTS` system table contains default parameter values. Query this table to determine default settings for all resource pools.
Default values specified in this table pertain only to user-defined resource pools. For built-in pool default values, see Built-In Pool Configuration.

<table>
<thead>
<tr>
<th>pool-name</th>
<th>The name of the resource pool. Built-in pool names cannot be used for user-defined pools.</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter-name</td>
<td>The parameter to set, listed below.</td>
</tr>
<tr>
<td>CASCADE TO</td>
<td>Specifies a secondary resource pool for executing queries that exceed the RUNTIMECAP setting of their assigned resource pool:</td>
</tr>
<tr>
<td></td>
<td>CASCADE TO secondary-pool</td>
</tr>
<tr>
<td>CPUAFFINITYMODE</td>
<td>Specifies whether the resource pool has exclusive or shared use of the CPUs specified in CPUAFFINITYSET:</td>
</tr>
<tr>
<td></td>
<td>CPUAFFINITYMODE {</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>• SHAREAD: Queries that run in this pool share its CPUAFFINITYSET CPUs with other Vertica resource pools.</td>
</tr>
<tr>
<td></td>
<td>• EXCLUSIVE: Duplicates CPUAFFINITYSET CPUs to this resource pool only, and excludes other Vertica resource pools. If CPUAFFINITYSET is set as a percentage, then that percentage of CPU resources available to Vertica is assigned solely for this resource pool.</td>
</tr>
<tr>
<td></td>
<td>• ANY (default): Queries in this resource pool can run on any CPU, invalid if CPUAFFINITYSET designates CPU resources.</td>
</tr>
<tr>
<td>CPUAFFINITYSET</td>
<td>Specifies which CPUs are available to this resource pool. All cluster nodes must have the same number of CPUs. The CPU resources assigned to this set are unavailable to general resource pools.</td>
</tr>
<tr>
<td></td>
<td>CPUAFFINITYSET {</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- `cpu-index[, ...]`: Dedicates one or more comma-delimited CPUs to this pool.

- `cpu-index_i-cpu-index_n`: Dedicates a range of contiguous CPU indexes to this pool.

- `integer%`: Percentage of all available CPUs to use for this pool. Vertica rounds this percentage down to include whole CPU units.

- `NONE` (default): No affinity set is assigned to this resource pool. The queries associated with this pool are executed on any CPU.

**EXECUTIONPARALLELISM**

Limits the number of threads used to process any single query issued in this resource pool.

**EXECUTIONPARALLELISM { integer | AUTO }**

- `integer`: A value between 1 and the number of cores. Setting this parameter to a reduced value increases throughput of short queries issued in the pool, especially if the queries are executed concurrently.

- `AUTO` (default): Vertica sets this value based on the number of cores, available memory, and amount of data in the system. Unless memory is limited, or the amount of data is very small, Vertica sets this value to the number of cores on the node.

**MAXCONCURRENCY**

Sets the maximum number of concurrent execution slots available to the resource pool, across the cluster:

**MAXCONCURRENCY { integer | NONE }**

`NONE` (default) specifies unlimited number of concurrent execution slots.

**MAXMEMORYSIZE**

The maximum size per node the resource pool can grow by borrowing memory from the **GENERAL** pool:

**MAXMEMORYSIZE**

```
(integer%)
(integer{K|M|G|T})
NONE
```
<table>
<thead>
<tr>
<th>MEMORYSIZE</th>
<th>The amount of total memory available to the Vertica resource manager that is allocated to this pool per node:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEMORYSIZE {</td>
</tr>
<tr>
<td></td>
<td>'integer%': Percentage of total memory</td>
</tr>
<tr>
<td></td>
<td>'integer{K</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: 0%. No memory allocated, the resource pool borrows memory from the <strong>GENERAL</strong> pool.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLANNEDCONCURRENCY</th>
<th>Specifies the preferred number queries to execute concurrently in the resource pool. This setting applies to the entire cluster:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PLANNEDCONCURRENCY { integer</td>
</tr>
<tr>
<td></td>
<td><strong>integer</strong>: The preferred number of concurrently executing queries. When possible, query resource budgets are limited to allow this level of concurrent execution.</td>
</tr>
<tr>
<td></td>
<td>AUTO (default): Value is calculated automatically at query runtime. Vertica sets this parameter to the lower of these two calculations, but never less than 4:</td>
</tr>
<tr>
<td></td>
<td>- Number of logical cores</td>
</tr>
<tr>
<td></td>
<td>- Memory divided by 2GB</td>
</tr>
</tbody>
</table>

For clusters where the number of logical cores differs
on different nodes, AUTO can apply differently on each node. Distributed queries run like the minimal effective planned concurrency. Single node queries run with the planned concurrency of the initiator.

Tip: Change this parameter only after evaluating performance over a period of time.

<table>
<thead>
<tr>
<th>PRIORITY</th>
<th>Specifies priority of queries in this pool when they compete for resources in the <strong>GENERAL</strong> pool:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY { integer</td>
<td>HOLD }</td>
</tr>
<tr>
<td>integer: A negative or positive integer value, where higher numbers denote higher priority:</td>
<td></td>
</tr>
<tr>
<td>User-defined pools: -100 to 100</td>
<td></td>
</tr>
<tr>
<td>Built-in pools <em>SYSQUERY</em>, <em>RECOVERY</em>, and <em>TM</em>: -110 to 110</td>
<td></td>
</tr>
<tr>
<td>HOLD: Sets priority to -999. Queries in this pool are queued until <strong>QUEUETIMEOUT</strong> is reached.</td>
<td></td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QUEUETIMEOUT</th>
<th>Specifies how long a request can wait for pool resources before it is rejected:</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUETIMEOUT { integer</td>
<td>NONE }</td>
</tr>
<tr>
<td>integer: Maximum wait time in seconds</td>
<td></td>
</tr>
<tr>
<td>NONE: No maximum wait time, request can be queued indefinitely.</td>
<td></td>
</tr>
<tr>
<td>Default: 300 seconds</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RUNTIMECAP</th>
<th>Prevents runaway queries by setting the maximum time a query in the pool can execute. If a query exceeds this setting, it tries to cascade to a secondary pool:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNTIMECAP { interval</td>
<td>NONE }</td>
</tr>
<tr>
<td>interval: An interval of 1 minute or 100 seconds; should not exceed one year.</td>
<td></td>
</tr>
<tr>
<td>NONE: No time limit on queries running in this pool.</td>
<td></td>
</tr>
</tbody>
</table>
To specify a value in days, provide an integer value. To provide a value less than one day, provide the interval in the format hours:minutes:seconds. For example a value of 1:30:00 would equal 90 minutes. If the user or session also has a RUNTIMECAP, the shorter limit applies.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>Determines how the resource manager should prioritize dedication of run-time resources (CPU, I/O bandwidth) to queries already running in this resource pool:</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>Specifies in seconds a time limit in which a query must finish before the resource manager assigns to it the resource pool's RUNTIMEPRIORITY. All queries begin running at a HIGH priority. When a query's duration exceeds this threshold, it is assigned the RUNTIMEPRIORITY of the resource pool.</td>
<td>2 seconds</td>
</tr>
</tbody>
</table>

### Privileges

The following parameters require superuser privileges on the resource pool:

- CASCADE TO
- CPUAFFINITYSET
- CPUAFFINITYMODE
- MAXMEMORYSIZE
- PRIORITY
- QUEUETIMEOUT

The following parameters require UPDATE privileges on the resource pool:
Examples

This example shows how to alter resource pool ceo_pool by setting the priority to 5.

```sql
=> ALTER RESOURCE POOL ceo_pool PRIORITY 5;
```

This example shows how to designate a secondary pool for the ceo_pool.

```sql
=> CREATE RESOURCE POOL second_pool;
=> ALTER RESOURCE POOL ceo_pool CASCADE TO second_pool;
```

See Also

- CREATE RESOURCE POOL
- CREATE USER
- DROP RESOURCE POOL
- RESOURCE_POOL_STATUS
- SET SESSION RESOURCE_POOL
- SET SESSION MEMORYCAP
- Managing Workloads

**ALTER ROLE RENAME**

Rename an existing role.
NOTE: You cannot use the ALTER ROLE RENAME command on a role added to the Vertica database with the LDAPLink service.

Syntax

```
ALTER ROLE name RENAME TO new_name;
```

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The current name of the role that you want to rename.</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_name</td>
<td>The new name for the role.</td>
</tr>
</tbody>
</table>

Privileges

Must be a superuser to rename a role.

Example

```
=> ALTER ROLE applicationadministrator RENAME TO appadmin;
ALTER ROLE
```

See Also

- CREATE ROLE
- DROP ROLE

**ALTER SCHEMA**

Changes one or more schemas in one of the following ways:
• Enables or disables inheritance of schema privileges by tables created in the schemas.

• Renames schemas.

Syntax

Set inheritance of schema privileges

ALTER SCHEMA [database.]schema[,...] DEFAULT {INCLUDE | EXCLUDE} SCHEMA PRIVILEGES

Rename schemas

ALTER SCHEMA [database.]schema[,...] RENAME TO new-schema-name[,...]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema to modify. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>DEFAULT {INCLUDE</td>
<td>EXCLUDE} SCHEMA PRIVILEGES</td>
</tr>
<tr>
<td>RENAME TO</td>
<td>Renames one or more schemas: RENAME TO new-schema-name[,...] The following requirements apply:</td>
</tr>
<tr>
<td></td>
<td>• The new schema name conforms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, models, and schemas in the database.</td>
</tr>
<tr>
<td></td>
<td>• If you specify multiple schemas to rename, the source and target lists must have the same number of names.</td>
</tr>
</tbody>
</table>
Caution: Renaming a schema referenced by a view causes the view to fail unless another schema is created to replace it.

Privileges

Superuser, or one of the following:

- To modify inheritance of privileges, schema owner or USAGE privilege on the schema
- To rename a schema, owner of the schema and CREATE privilege on the database

Swapping Schemas

Renaming schemas is useful for swapping schemas without actually moving data. To facilitate the swap, enter a non-existent, temporary placeholder schema. For example, the following ALTER SCHEMA statement uses the temporary schema temps to facilitate swapping schema S1 with schema S2. In this example, S1 is renamed to temps. Then S2 is renamed to S1. Finally, temps is renamed to S2.

```
ALTER SCHEMA S1, S2, temps RENAME TO temps, S1, S2;
```

Examples

The following example renames schemas S1 and S2 to S3 and S4, respectively:

```
=> ALTER SCHEMA S1, S2 RENAME TO S3, S4;
```

This example sets the default behavior for new table t2 to automatically inherit the schema's privileges:

```
=> ALTER SCHEMA s1 DEFAULT INCLUDE SCHEMA PRIVILEGES;
=> CREATE TABLE s1.t2 (i, int);
```

This example sets the default for new tables to not automatically inherit privileges from the schema:

```
=> ALTER SCHEMA s1 DEFAULT EXCLUDE SCHEMA PRIVILEGES;
```
See Also

- CREATE SCHEMA
- DROP SCHEMA

**ALTER SEQUENCE**

Changes a named sequence in two ways:

- Sets parameters that control sequence behavior—for example, its start value, and range of minimum and maximum values. These changes take effect only when you start a new database session.

- Sets sequence name, schema, or ownership. These changes take effect immediately.

Note: You can only modify a named sequence—that is, a sequence that was defined by CREATE SEQUENCE. AUTO_INCREMENT and IDENTITY sequences are owned by the table where they were created, and cannot be changed independently of that table.

**Syntax**

Change sequence behavior:

```sql
ALTER SEQUENCE [[database.]schema.]sequence
... [ INCREMENT [ BY ] integer ]
... [ MINVALUE integer | NO MINVALUE ]
... [ MAXVALUE integer | NO MAXVALUE ]
... [ RESTART [ WITH ] integer ]
... [ CACHE integer | NO CACHE ]
... [ CYCLE | NO CYCLE ]
```

Change sequence name, schema, or ownership:

```sql
ALTER SEQUENCE [schema.]sequence-name {
... RENAME TO seq-name
... | SET SCHEMA schema-name
... | OWNER TO owner-name
}
```
### Parameters

<table>
<thead>
<tr>
<th><strong>schema</strong></th>
<th>Specifies a schema, by default public. If <em>schema</em> is any schema other than public, you must supply the schema name. For example:</th>
<th></th>
</tr>
</thead>
</table>
| myschema.thisDbObject | If you specify a database, it must be the current database.  
If you do not specify a schema, the table is created in the default schema. |  |

<table>
<thead>
<tr>
<th><strong>sequence</strong></th>
<th>The name of the sequence to alter.</th>
<th></th>
</tr>
</thead>
</table>

| **INCREMENT [BY] integer** | A positive or negative integer that specifies how much to increment or decrement the sequence on each call to `NEXTVAL`, by default set to 1.  
**Note:** Setting this parameter to `integer` guarantees that column values always increment by at least `integer`. However, column values can sometimes increment by more than `integer` unless you also set the `NO CACHE` parameter. |  |

| **MINVALUE integer** (default) | Modifies the minimum value a sequence can generate. If you change this value and the current value exceeds the range, the current value is changed to the minimum value if increment is greater than zero, or to the maximum value if increment is less than zero.  
**NO MINVALUE** |  |

| **MAXVALUE integer** (default) | Modifies the maximum value for the sequence. If you change this value and the current value exceeds the range, the current value is changed to the minimum value if increment is greater than zero, or to the maximum value if increment is less than zero. |  |

| **RESTART [WITH] integer** | Changes the current value of the sequence to `integer`. The next call to `NEXTVAL` returns `integer`.  
**Caution:** Using `ALTER SEQUENCE` to set a sequence start value below its current value can result in duplicate keys. |  |

| **CACHE integer** (default) | Specifies how many sequence numbers are pre-allocated and stored in memory for faster access. Vertica sets up caching for each session, and distributes it across all nodes. By default, the sequence cache is |  |
set to 250,000.
For details, see Distributing Named Sequences in the Administrator's Guide.

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>NO CYCLE (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies whether the sequence can wrap when its minimum or maximum values are reached:</td>
</tr>
<tr>
<td></td>
<td>• CYCLE: The sequence wraps as follows:</td>
</tr>
<tr>
<td></td>
<td>■ When an incrementing sequence reaches its upper limit, it is reset to its minimum value.</td>
</tr>
<tr>
<td></td>
<td>■ When an decrementing sequence reaches its lower limit, it is reset to its maximum value.</td>
</tr>
<tr>
<td></td>
<td>• NO CYCLE (default): Calls to NEXTVAL return an error after the sequence reaches its maximum or minimum value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RENAME TO</th>
<th>seq-name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Renames a sequence within the current schema, where seq-name conforms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SET SCHEMA</th>
<th>schema-name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Moves the sequence to schema schema-name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OWNER TO</th>
<th>owner-name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reassigns the current sequence owner to the specified owner.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser, or sequence owner with the following privileges:

- Rename sequence: USAGE and CREATE privileges on sequence.
- Move to another schema: USAGE privileges on current schema, and CREATE privilege on destination schema.
- Reassign ownership: Current sequence owner.

**Examples**

See Altering Sequences in the Administrator's Guide.
See Also

**CREATE SEQUENCE**

**ALTER SESSION**

Use `ALTER SESSION` to set and clear session-level configuration parameter values for the current session.

**Syntax**

```
ALTER SESSION {
    SET [PARAMETER] parameter-name=value [, ...]
    | CLEAR [PARAMETER] parameter-name[, ...]
    | SET UDPARAMETER [ FOR { namespace | libname } ] key=value [, ...]
    | CLEAR UDPARAMETER [ FOR { namespace | libname } ] key [, ...]
}
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET [PARAMETER]</td>
<td>Sets one or more configuration parameters to the specified value.</td>
</tr>
<tr>
<td>CLEAR [PARAMETER]</td>
<td>Clears one or more specified configuration parameters.</td>
</tr>
<tr>
<td>SET UDPARAMETER</td>
<td>Sets a user-defined session parameter, can be used in combination with a UDx. If omit a name space, Vertica uses the PUBLIC name space.</td>
</tr>
<tr>
<td></td>
<td><strong>Key size values:</strong></td>
</tr>
<tr>
<td></td>
<td>• Set from client side: 128 characters</td>
</tr>
<tr>
<td></td>
<td>• Set from UDx side: Unlimited</td>
</tr>
<tr>
<td>CLEAR UDPARAMETER</td>
<td>Clears one or more user-defined session parameters.</td>
</tr>
</tbody>
</table>
Privileges

None

Examples

Set and clear a parameter

- Force all UDxes that support fenced mode to run in fenced mode, even if their definition specified is NOT FENCED:

  => ALTER SESSION SET ForceUDxFencedMode = 1;

- Clear ForceUDxFencedMode at the session level. The value now reflects the default value (0).

  => ALTER SESSION CLEAR ForceUDxFencedMode;

Set and clear a user-defined parameter

- Set the value of RowCount, in the MyLibrary library and PUBLIC namespace, to 25.

  => ALTER SESSION SET UDPARAMETER FOR MyLibrary RowCount = 25;

- Clear RowCount at the session level. The value now reflects the default value (0).

  => ALTER SESSION CLEAR UDPARAMETER FOR MyLibrary RowCount;

See Also

SESSION_PARAMETERS
**ALTER SUBNET**

Renames an existing subnet.

**Syntax**

```
ALTER SUBNET subnet-name RENAME TO new-subnet-name
```

**Parameters**

<table>
<thead>
<tr>
<th>subnet-name</th>
<th>The name of the existing subnet.</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-subnet-name</td>
<td>The new name for the subnet.</td>
</tr>
</tbody>
</table>

**Privileges**

Must be a superuser to alter a subnet.

**Examples**

This example renames a subnet.

```
=> ALTER SUBNET mysubnet RENAME TO myNewSubnet;
```

**ALTER TABLE**

Modifies the metadata of an existing table. The changes are auto-committed.

You cannot modify the metadata of an anchor table for a live aggregate projection or Top-K projection.
General Usage

ALTER TABLE [[database.]schema.]table {
  ... ADD COLUMN column data-type
      [ column-constraint ]
      [ ENCODING encoding-type ]
      [ RESTRICT | CASCADE ]
      [ PROJECTIONS (projection [, ...]) ]
  ... | ADD table-constraint
  ... | ALTER COLUMN column column-setting
  ... | ALTER CONSTRAINT constraint-name { ENABLED | DISABLED }
  ... | DROP CONSTRAINT constraint-name [ CASCADE | RESTRICT ]
  ... | DROP [ COLUMN ] column [ CASCADE | RESTRICT ]
  ... | FORCE OUTER integer
  ... | { INCLUDE | EXCLUDE | MATERIALIZE } [ SCHEMA ] PRIVILEGES
  ... | OWNER TO owner
  ... | partition-clause
  ... | REMOVE PARTITIONING
  ... | RENAME [ COLUMN ] name TO new-name
  ... | REORGANIZE
  ... | SET DATA TYPE datatype
  ... | SET SCHEMA schema
  ... | SET STORAGE load-option
}

Table Renaming

ALTER TABLE [[database.]schema.]table[,]... RENAME TO new-table-name[,]...

Parameters

<table>
<thead>
<tr>
<th>[ database .]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

| table               | The table to alter.                                                                                                                                                                             |

<table>
<thead>
<tr>
<th>ADD COLUMN</th>
<th>Adds a column to the table and to its superprojections:</th>
</tr>
</thead>
</table>
|                     | ADD COLUMN column-name datatype
      [column-constraint]
      [ENCODING encoding-type]
      [RESTRICT | CASCADE]
      [PROJECTIONS (projection [, ...]) ]

You can qualify the new column definition with one of these options:

- column-constraint specifies a column constraint as follows:
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD table-constraint</td>
<td>Adds a constraint to a table that does not have any associated projections. See About Constraints in the Administrator's Guide.</td>
</tr>
</tbody>
</table>
| ALTER COLUMN | Alters a setting for column:  
| column-setting | column-setting can be one of the following:  
| SET DEFAULT expression | sets column values to the specified expression.  
| DROP DEFAULT | drops the DEFAULT setting.  
| SET USING expression | specifies to set this column's values from the result set returned by expression. When you make this change, Vertica automatically calls the function REFRESH_COLUMNS, which populates the column from the result set |

**ENCODING** specifies the column's encoding type, by default set to AUTO.

**RESTRICT** (default) adds the new column to pre-join projections only if they are anchored to the updated table. **CASCADE** updates all pre-join projections where the table is specified, regardless of whether they are anchored to it.

**PROJECTIONS** specifies to add the new column to one or more projections, where projection is the projection base name and is not a superprojection. Vertica adds the column to all buddies of each projection.

Note: Altering an existing table column to specify a DEFAULT expression has no effect on existing values in that column. Vertica applies the DEFAULT expression only on new rows when they are added to the table, through load operations such as INSERT and COPY. To refresh the entire column with its DEFAULT expression, update the column as follows:

```
UPDATE table-name SET column-name=DEFAULT;
```
<table>
<thead>
<tr>
<th><strong>ALTER CONSTRAINT</strong></th>
<th>Applies to automatic enforcement of primary key, unique key, and check constraints:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER CONSTRAINT <strong>constraint-name</strong> {ENABLED</td>
<td>DISABLED}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DROP CONSTRAINT</strong></th>
<th>Drops the specified table constraint from the table:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP CONSTRAINT <strong>constraint-name</strong> [CASCADE</td>
<td>RESTRICT]</td>
</tr>
</tbody>
</table>

You can qualify DROP CONSTRAINT with one of these options:

- **CASCADE**: Drops a constraint and all dependencies in other tables.
- **RESTRICT**: Does not drop a constraint if there are dependent objects. Same as the default behavior.

Dropping a table constraint has no effect on views that reference the table.

<table>
<thead>
<tr>
<th><strong>DROP [COLUMN]</strong></th>
<th>Drops the specified column from the table and that column's ROS containers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP [COLUMN] <strong>column</strong> [CASCADE</td>
<td>RESTRICT]</td>
</tr>
</tbody>
</table>
You can qualify `DROP COLUMN` with one of these options:

- **CASCADE** is required if the table column to drop has dependencies.
- **RESTRICT** drops the column only from the given table.

See [Dropping Table Columns](#) in the Administrator's Guide.

<table>
<thead>
<tr>
<th>FORCE OUTER integer</th>
<th>Specifies whether a table is joined to another as an inner or outer input. For details, see <a href="#">Controlling Join Inputs</a> in Analyzing Data.</th>
</tr>
</thead>
</table>
| `{ INCLUDE | EXCLUDE | MATERIALIZE } [SCHEMA] PRIVILEGES` | Specifies default inheritance of schema privileges for this table:  
  
  - **EXCLUDE [SCHEMA] PRIVILEGES** (default) disables inheritance of privileges from the schema.  
  
  - **INCLUDE [SCHEMA] PRIVILEGES** grants the table the same privileges granted to its schema.  
  
  - **MATERIALIZE**: Copies grants to the table and creates a GRANT object on the table. This disables the inherited privileges flag on the table, so you can:  
    - Grant more specific privileges at the table level  
    - Use schema-level privileges as a template  
    - Move the table to a different schema  
    - Change schema privileges without affecting the table  

  **Note:** If inherited privileges are disabled at the database level, schema privileges can still be materialized.

  For more information see [Grant Inherited Privileges](#) in the Administrator's Guide.

<table>
<thead>
<tr>
<th>OWNER TO owner</th>
<th>Changes the table owner. See <a href="#">Changing Table Ownership</a> in the Administrator's Guide.</th>
</tr>
</thead>
</table>
| partition-clause | Invalid for external tables, specifies how to repartition table data storage as follows:  

  ```sql  
  PARTITION BY partition-expression [ GROUP BY group-expression ] [ REORGANIZE ]  
  ```  

  For more information see [Controlling Partitioning](#) in the Administrator's Guide.
For details about partition and group expressions, see [Partition Clause](#).

If the partition clause includes REORGANIZE and the table previously specified no partitioning, the VerticaTuple Mover immediately implements the partition clause. If the table previously specified partitioning, Vertica compares the new and previous partition clauses:

- If the clauses are identical, Vertica ignores the REORGANIZE keyword.
- If the clauses are different, Vertica immediately implements the new partition clause: the Tuple Mover evaluates current partitioning and reorganizes ROS storage containers as needed to conform with the new partitioning policy.

<table>
<thead>
<tr>
<th>REMOVE PARTITIONING</th>
<th>Specifies to remove partitioning from a table definition. The Tuple Mover subsequently removes existing partitions from ROS containers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME [COLUMN]</td>
<td>Renames the specified column within the table. See <a href="#">Renaming Columns</a> in the Administrator's Guide.</td>
</tr>
<tr>
<td>REORGANIZE</td>
<td>If used alone with ALTER TABLE, REORGANIZE invokes the Tuple Mover to reorganize ROS storage containers as needed to conform with the table’s current partitioning policy. ALTER TABLE...REORGANIZE and Vertica meta-function PARTITION_TABLE operate identically. REORGANIZE can also qualify a new partition clause.</td>
</tr>
<tr>
<td>SET DATA TYPE <em>datatype</em></td>
<td>Changes the column's data type to any type whose conversion does not require storage reorganization. See <a href="#">Changing a Column Data Type</a> in the Administrator's Guide.</td>
</tr>
<tr>
<td>SET SCHEMA</td>
<td>Moves the table from one schema to another. Vertica automatically moves all projections that are anchored to the source table to the destination schema. It also moves all IDENTITY and AUTO_INCREMENT columns to the destination schema. For details, see <a href="#">Moving Tables to Another Schema</a> in the Administrator's Guide.</td>
</tr>
<tr>
<td>SET STORAGE <em>Load-method</em></td>
<td>Specifies default load behavior for all DML operations on this table, such as INSERT and COPY, one of the following:</td>
</tr>
</tbody>
</table>
- **AUTO (default):** Initially loads data into WOS, suitable for smaller bulk loads.
- **DIRECT:** Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.
- **TRICKLE:** Loads data only into WOS, suitable for frequent incremental loads.

For details, see [Choosing a Load Method](#) in the Administrator's Guide.

<table>
<thead>
<tr>
<th>RENAME TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renames one or more tables:</td>
</tr>
<tr>
<td>RENAME TO new-table-name[, ...]</td>
</tr>
</tbody>
</table>

The following requirements apply:

- The new table name conforms to conventions described in [Identifiers](#). It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.
- If you specify multiple tables to rename, the source and target lists must have the same number of names.
- Renaming a table requires USAGE and CREATE privileges on the schema that contains it.

**Privileges**

The following privileges are required:

- Table owner or superuser, and USAGE privileges on the table schema to make the following changes:
  - Add, drop, rename, or alter column.
  - Add or drop a constraint.
  - Partition or repartition the table.

Renaming a table requires USAGE and CREATE privilege on the table schema.

Moving a table to a new schema requires:
• USAGE privilege on the old schema

• CREATE privilege on new schema

Locked Tables

If the operation cannot obtain an O lock on the target table, Vertica tries to close any internal Tuple Mover sessions that are running on that table. If successful, the operation can proceed. Explicit Tuple Mover operations that are running in user sessions do not close. If an explicit Tuple Mover operation is running on the table, the operation proceeds only when the operation is complete.

See Also

• Managing Tables

• Altering Table Definitions

• Adding Table Columns

Table-Constraint

Adds a constraint to table metadata. You can specify table constraints with CREATE TABLE, or add a constraint to an existing table with ALTER TABLE. For details, see Adding Constraints in the Administrator's Guide.

Note: Adding a constraint to a table that is referenced in a view does not affect the view.

Syntax

[ CONSTRAINT constraint-name ]
{
  ... PRIMARY KEY (column[,...]) [ ENABLED | DISABLED ]
  ... FOREIGN KEY (column[,...]) REFERENCES table [ (column[,...]) ]
  ... UNIQUE (column[,...]) [ ENABLED | DISABLED ]
  ... CHECK (expression) [ ENABLED | DISABLED ]
}
## Parameters

<table>
<thead>
<tr>
<th><strong>CONSTRAINT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>constraint-name</strong></td>
</tr>
<tr>
<td>Assigns a name to the constraint. Vertica recommends that you name all constraints.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRIMARY KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines one or more NOT NULL columns as the primary key as follows:</td>
</tr>
<tr>
<td>PRIMARY KEY (<em>column[,...]</em>) [ ENABLED</td>
</tr>
<tr>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See <strong>Enforcing Constraints</strong> below.</td>
</tr>
<tr>
<td>If you do not name a primary key constraint, Vertica assigns the name <strong>C_PRIMARY</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOREIGN KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adds a referential integrity constraint defining one or more columns as foreign keys as follows:</td>
</tr>
<tr>
<td>FOREIGN KEY (<em>column[,...]</em>) REFERENCES <em>table</em> [(column[,...])]</td>
</tr>
<tr>
<td>If you omit <em>column</em> references, If you omit <em>column</em>, Vertica references the primary key in <em>table</em>.</td>
</tr>
<tr>
<td>If you do not name a foreign key constraint, Vertica assigns the name <strong>C_FOREIGN</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNIQUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies that the data in a column or group of columns is unique with respect to all table rows, as follows:</td>
</tr>
<tr>
<td>UNIQUE (<em>column[,...]</em>) [ENABLED</td>
</tr>
<tr>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See <strong>Enforcing Constraints</strong> below.</td>
</tr>
<tr>
<td>If you do not name a unique constraint, Vertica assigns the name <strong>C_UNIQUE</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHECK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies a check condition as an expression that returns a Boolean value, as follows:</td>
</tr>
<tr>
<td>CHECK (<em>expression</em>) [ENABLED</td>
</tr>
<tr>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See <strong>Enforcing Constraints</strong> below.</td>
</tr>
</tbody>
</table>
If you do not name a check constraint, Vertica assigns the name C_CHECK.

Privileges

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

Enforcing Constraints

A table can specify whether Vertica automatically enforces a primary key, unique key or check constraint with the keyword ENABLED or DISABLED. If you omit ENABLED or DISABLED, Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- EnableNewPrimaryKeysByDefault
- EnableNewUniqueKeysByDefault
- EnableNewCheckConstraintsByDefault

For details, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

Examples

The following example creates a table (t01) with a primary key constraint.

```
CREATE TABLE t01 (id int CONSTRAINT sampleconstraint PRIMARY KEY);
```

This example creates the same table without the constraint, and then adds the constraint with

```
ALTER TABLE t01 ADD CONSTRAINT
```

```
CREATE TABLE t01 (id int);
CREATE TABLE
```

```
ALTER TABLE t01 ADD CONSTRAINT sampleconstraint PRIMARY KEY(id);
WARNING 2623: Column "id" definition changed to NOT NULL
```
The following example creates a table (addapk) with two columns, adds a third column to the table, and then adds a primary key constraint on the third column.

```sql
=> CREATE TABLE addapk (col1 INT, col2 INT);
CREATE TABLE

=> ALTER TABLE addapk ADD COLUMN col3 INT;
ALTER TABLE

=> ALTER TABLE addapk ADD CONSTRAINT col3constraint PRIMARY KEY (col3) ENABLED;
WARNING 2623: Column "col3" definition changed to NOT NULL
ALTER TABLE
```

Using the sample table addapk, check that the primary key constraint is enabled (is_enabled is t).

```sql
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col3constraint</td>
<td>col3</td>
<td>p</td>
<td>t</td>
</tr>
</tbody>
</table>
(1 row)
```

This example disables the constraint using ALTER TABLE ALTER CONSTRAINT.

```sql
=> ALTER TABLE addapk ALTER CONSTRAINT col3constraint DISABLED;
```

Check that the primary key is now disabled (is_enabled is f).

```sql
=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col3constraint</td>
<td>col3</td>
<td>p</td>
<td>f</td>
</tr>
</tbody>
</table>
(1 row)
```

For a general discussion of constraints, see About Constraints. For additional examples of creating and naming constraints, see Naming Constraints.

---

**ALTER USER**

Changes a database user account. Changes that you make to a user account affect only future user sessions.
Important: The following ALTER USER parameters are invalid for a user who is added to the Vertica database with the LDAPLink service:

- IDENTIFIED BY
- PROFILE
- SECURITY ALGORITHM

Syntax

ALTER USER name user-parameter setting[, ...]

user-parameter

ACCOUNT
DEFAULT ROLE*
GRACEPERIOD
IDENTIFIED BY
IDLESESSIONTIMEOUT
MAXCONNECTIONS
MEMORYCAP
PASSWORD EXPIRE
PROFILE
RENAME TO*
RESOURCE POOL
RUNTIMECAP
SEARCH_PATH
SECURITY_ALGORITHM
TEMPSPACECAP

* Cannot be used in combination with other parameters

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Specifies the name of the new user. Names that contain special characters must be double-quoted. To enforce case-sensitivity, use double-quotes. For details on name requirements, see Creating a Database Name and Password.</th>
</tr>
</thead>
</table>
| ACCOUNT { LOCK | UNLOCK } | Locks or unlocks a user's access to the database:  
- UNLOCK (default)  
- LOCK prevents a new user from logging in. This can be useful |
when creating an account for a user who does not need immediate access.

Tip: To automate account locking, set a maximum number of failed login attempts with `CREATE PROFILE`.

| **DEFAULT ROLE**<br>`roles-expression` | Specifies what roles are the default roles for this user, with one of the following expressions:
| | - **NONE** (default): Removes all default roles.
| | - A comma-delimited list of roles.
| | - **ALL**: Sets as default all user roles.
| | - **ALL EXCEPT** `role[, ...]`: A comma-delimited list of roles to exclude as default roles.
| | Default roles are automatically activated when a user logs in. The roles specified by this parameter supersede any roles assigned earlier.
| | **Note:** `DEFAULT ROLE` cannot be specified in combination with other `ALTER USER` parameters.

| **GRACEPERIOD**<br>`limit` | Specifies how long a user query can block on any session socket, where `limit` is one of the following:
| | - **NONE** (default): Removes any grace period previously set on session queries.
| | - `'interval'`: Specifies as an `interval` the maximum grace period for current session queries, up to 20 days.
| | For details, see [Handling Session Socket Blocking](#).

| **IDENTIFIED BY**<br>`'[new-password]'`<br>[REPLACE `'current-password'`] | Sets a new password for the user, where `new-password` must conform to the password complexity policy set by the user's profile.
| | Superusers can change the password for any user, and are not required to specify the `REPLACE` clause. Non-superusers can only change their own password, and must supply their current password with the `REPLACE` clause.
If you supply an empty string, the user's current password is removed, and the user is no longer prompted for a password when starting a new session.

For details, see Password Guidelines and Creating a Database Name and Password.

**IDLESESSIONTIMEOUT**

The length of time the system waits before disconnecting an idle session, where `limit` is one of the following:

- NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user.
- `'interval'`: An interval value, up to one year.

For details, see Managing Client Connections.

**MAXCONNECTIONS**

Indicates the maximum number of connections the user can have to the server, where `limit` is one of the following:

- NONE (default): No limit set. If you omit this parameter, the user can have an unlimited number of connections.
- `'integer'` ON NODE: Sets the maximum number of connections to each node to `integer`.
- `'integer'` ON DATABASE: Sets the maximum number of connections across the database cluster to `integer`.

For details, see Managing Client Connections.

**MEMORYCAP**

Specifies how much memory can be allocated to user requests, where `limit` is specified in this format:

- NONE (default): No limit
- `'max-expression'`: A string value that specifies the memory limit, one of the following:
  - `int%` — Expresses the maximum as a percentage of total memory available to the Resource Manager, where `int` is an integer value between 0 and 100. For example:
    
    MEMORYCAP '40%'
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`int{K</td>
<td>M</td>
</tr>
<tr>
<td>PASSWORD EXPIRE</td>
<td>Forces immediate expiration of the user's password. The user must change the password on the next login.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Forc sức immediate expiration of the user's password. The user must change the password on the next login.</td>
</tr>
<tr>
<td>Note: PASSWORD EXPIRE has no effect when using external password authentication methods such as LDAP or Kerberos.</td>
<td></td>
</tr>
<tr>
<td>PROFILE <code>profile</code></td>
<td>Assigns a profile that controls password requirements for this user, where <code>profile</code> is one of the following:</td>
</tr>
<tr>
<td><code>DEFAULT</code> (default): Assigns the default database profile to this user.</td>
<td></td>
</tr>
<tr>
<td><code>profile-name</code>: A profile that is defined by CREATE PROFILE.</td>
<td></td>
</tr>
<tr>
<td>RENAME TO <code>new-user-name</code></td>
<td>Assigns the user a new user name. All privileges assigned to the user remain unchanged.</td>
</tr>
<tr>
<td>Note: RENAME TO cannot be specified in combination with other ALTER USER parameters.</td>
<td></td>
</tr>
<tr>
<td>RESOURCE POOL <code>pool-name</code></td>
<td>Assigns a default resource pool to this user. The user must also be granted privileges to this pool, unless privileges to the pool are set to PUBLIC.</td>
</tr>
<tr>
<td>RUNTIMECAP <code>limit</code></td>
<td>Specifies how long this user's queries can execute, where <code>limit</code> is one of the following:</td>
</tr>
<tr>
<td>NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user.</td>
<td></td>
</tr>
<tr>
<td>'interval' An interval value, up to one year.</td>
<td></td>
</tr>
<tr>
<td>A query's runtime limit can be set at three levels: the user's runtime limit, the user's resource pool, and the session setting. For more information, see Setting a Runtime Limit for Queries in the Administrator's Guide.</td>
<td></td>
</tr>
</tbody>
</table>
### SEARCH_PATH `path`

Specifies the user's default search path that tells Vertica which schemas to search for unqualified references to tables and UDFs, where `path` is one of the following:

- **DEFAULT** (default): Sets the search path as follows:
  
  ```
  "$user", public, v_catalog, v_monitor, v_internal
  ```

- A comma-delimited list of schemas.

For details, see [Setting Search Paths](#) in the Administrator's Guide.

### SECURITY_ALGORITHM `'algorithm'`

Set the user-level security algorithm for hash authentication, where `algorithm` is one of the following:

- **NONE** (default): Uses the MD5 algorithm for hash authentication.
- **MD5**
- **SHA512**

The user's password expires when you change the `SECURITY_ALGORITHM` value, and must be reset.

### TEMPSPACECAP `limit`

Limits how much temporary file storage is available for user requests, where `limit` is one of the following:

- **NONE** (default): No limit

- **`max-expression`**: A string value that specifies the storage limit, one of the following:
  
  - `int%` — Expresses storage as a percentage of total file space is available, where `int` is an integer value between 0 and 100. For example:
    
    ```
    TEMPSPACECAP '40%'
    ```

  - `int{K|M|G|T}` — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example:
    
    ```
    TEMPSPACECAP '10G'
    ```
Privileges

- Superuser

- Non-superusers can change their own user accounts with these options:
  - IDENTIFIED BY
  - RESOURCE POOL
  - SEARCH_PATH
  - SECURITY_ALGORITHM

Examples

Change the user password

```
=> CREATE USER user1;
=> ALTER USER user1 IDENTIFIED BY 'newpassword';
```

Change the security algorithm and password

This example changes the user's hash authentication and password to SHA-512 and newpassword, respectively. When you execute the ALTER USER statement, Vertica hashes the password, using the SHA-512 algorithm, and saves the hashed version:

```
=> CREATE USER user1;
=> ALTER USER user1 SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword'
```

Assign user default roles

This example makes a user's assigned roles the user's default roles. The first ALTER USER statement makes role1 the default role. The second ALTER USER statement makes role1, role2, and role3 the user's default roles.

```
=> CREATE USER user1;
CREATE USER
=> GRANT role1, role2, role3 to user1;
=> ALTER USER user1 default role role1;
=> ALTER USER user1 default role ALL;
```

Assign user default roles with EXCEPT

This example makes all the user's assigned roles default roles with the exception of role1.

```
=> CREATE USER user1;
CREATE USER
=> GRANT role1, role2, role3 to user1;
=> ALTER USER user1 default role role1;
=> ALTER USER user1 default role ALL;
```
See Also

- **CREATE USER**
- **DROP USER**

**ALTER VIEW**

Modifies the metadata of an existing view. The changes are auto-committed.

**Syntax**

**General Usage**

```
ALTER VIEW [database.]schema.]view {
... | OWNER TO owner
... | SET SCHEMA schema
... | { INCLUDE | EXCLUDE | MATERIALIZE } [ SCHEMA ] PRIVILEGES
```

**View Renaming**

```
ALTER VIEW [database.]schema.]view[,...] RENAME TO new-view-name[,,...]
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>[database.]schema</strong></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>myschema.thisDbObject</code></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>view</strong></th>
<th>The view to alter.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SET SCHEMA schema</strong></td>
<td>Moves the view from one schema to another.</td>
</tr>
<tr>
<td>OWNER TO owner</td>
<td>Changes the view owner.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>{ INCLUDE</td>
<td>EXCLUDE</td>
</tr>
<tr>
<td>MATERIALIZE }</td>
<td>[SCHEMA] PRIVILEGES</td>
</tr>
</tbody>
</table>

Specifies default inheritance of schema privileges for this view:

- **EXCLUDE [SCHEMA] PRIVILEGES** (default) disables inheritance of privileges from the schema.

- **INCLUDE [SCHEMA] PRIVILEGES** grants the view the same privileges granted to its schema.

- **MATERIALIZE**: Copies grants to the view and creates a GRANT object on the view. This disables the inherited privileges flag on the view, so you can:
  - Grant more specific privileges at the view level
  - Use schema-level privileges as a template
  - Move the view to a different schema
  - Change schema privileges without affecting the view

**Note:** If inherited privileges are disabled at the database level, schema privileges can still be materialized.

For more information see [Grant Inherited Privileges](#).

<table>
<thead>
<tr>
<th>RENAME TO</th>
<th>Renames one or more views:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENAME TO new-view-name[,...]</td>
<td>The following requirements apply:</td>
</tr>
</tbody>
</table>

- The new view name conforms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.

- If you specify multiple views to rename, the source and target lists must have the same number of names.

- Renaming a view requires USAGE and CREATE privileges on the schema that contains the view.
Privileges
Changing a view requires the following privileges:

- Superuser
- If renaming a view, CREATE privileges on the schema in which the view is renamed.

Example
The following command renames view1 to view2:

```sql
=> CREATE VIEW view1 AS SELECT * FROM t;
CREATE VIEW
=> ALTER VIEW view1 RENAME TO view2;
ALTER VIEW
```

BEGIN
Starts a transaction block. BEGIN is a synonym for START TRANSACTION.

Syntax
BEGIN [ WORK | TRANSACTION ] [ isolation-level ] [ transaction-mode]
and where transaction_mode is one of:
READ { ONLY | WRITE }

Parameters

<table>
<thead>
<tr>
<th>WORK</th>
<th>TRANSACTION</th>
<th>Optional keywords for readability purposes only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>isolation-level</td>
<td>Specifies the transaction's isolation level, which determines what data the transaction can access when other transactions are running concurrently. You can set isolation-level to one of the following:</td>
<td></td>
</tr>
</tbody>
</table>

- **READ COMMITTED** (default)
- **SERIALIZABLE**
- **REPEATABLE READ** (automatically converted to SERIALIZABLE)
- **READ UNCOMMITTED** (automatically converted to READ COMMITTED)

For detailed information, see Transactions in Vertica Concepts.

<table>
<thead>
<tr>
<th>transaction-mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ { ONLY</td>
</tr>
<tr>
<td>Transaction mode can be one of the following:</td>
</tr>
<tr>
<td>- READ WRITE—(default) The transaction is read/write.</td>
</tr>
<tr>
<td>- READ ONLY—The transaction is read-only.</td>
</tr>
</tbody>
</table>

Setting the transaction session mode to read-only disallows the following SQL commands, but does not prevent all disk write operations:

- INSERT, UPDATE, DELETE, and COPY if the table they would write to is not a temporary table
- All CREATE, ALTER, and DROP commands
- GRANT, REVOKE, and EXPLAIN if the command it would run is among those listed.

**Privileges**

None

**Examples**

This example shows how to begin a transaction and set the isolation level.
=> BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
BEGIN
=> CREATE TABLE sample_table (a INT);
CREATE TABLE
=> INSERT INTO sample_table (a) VALUES (1);
OUTPUT
--------
1
(1 row)

See Also

- Transactions
- Creating Transactions
- COMMIT
- END
- ROLLBACK
COMMENT ON Statements

The following functions allow you to create comments associated with Vertica database objects:

- COMMENT ON COLUMN
- COMMENT ON CONSTRAINT
- COMMENT ON FUNCTION
- COMMENT ON LIBRARY
- COMMENT ON NODE
- COMMENT ON PROJECTION
- COMMENT ON SCHEMA
- COMMENT ON SEQUENCE
- COMMENT ON TABLE
- COMMENT ON TRANSFORM FUNCTION
- COMMENT ON VIEW

COMMENT ON COLUMN

Adds, revises, or removes a projection column comment. You can only add comments to projection columns, not to table columns. Each object can have one comment. Comments are stored in the system table COMMENTS.

Syntax

```
COMMENT ON COLUMN [[database.]schema.]projection.column IS {'comment' | NULL}
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code>&lt;br&gt; If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>projection.column</code></td>
<td>The name of the projection and column with which to associate the comment.</td>
</tr>
<tr>
<td><code>comment</code></td>
<td>Specifies the comment text to add. If a comment already exists for this column, this comment overwrites the previous comment. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. You can enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td><code>NULL</code></td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Example

The following example adds a comment to the `customer_name` column in the `customer_dimension` projection:

```
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS 'Last name only';
```
The following examples remove a comment from the customer_name column in the customer_dimension projection in two ways, using the NULL option, or specifying a blank string:

```sql
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS NULL;
=> COMMENT ON COLUMN customer_dimension_vmart_node01.customer_name IS ';
```

### COMMENT ON CONSTRAINT

Adds, revises, or removes a comment on a constraint. Each object can have one comment. Comments are stored in the system table `COMMENTS`.

#### Syntax

```
COMMENT ON CONSTRAINT constraint ON [[database.]schema.]table IS ... {'comment' | NULL};
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>constraint</code></td>
<td>The name of the constraint associated with the comment.</td>
</tr>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema&gt;thisDbObject</code></td>
</tr>
<tr>
<td><code>table</code></td>
<td>The name of the table constraint with which to associate a comment.</td>
</tr>
<tr>
<td><code>comment</code></td>
<td>Specifies the comment text to add. If a comment already exists for this constraint, this comment overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. You can enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td><code>NULL</code></td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>
Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Example

The following example adds a comment to the constraint_x constraint on the promotion_dimension table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS 'Primary key';
```

The following examples remove a comment from the constraint_x constraint on the promotion_dimension table:

```
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS NULL;
=> COMMENT ON CONSTRAINT constraint_x ON promotion_dimension IS '';
```

COMMENT ON FUNCTION

Adds, revises, or removes a comment on a function. Each object can have one comment. Comments are stored in the system table COMMENTS.

Syntax

```
COMMENT ON FUNCTION [[database.]schema.]function (function-args) IS { 'comment' | NULL };
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject</td>
</tr>
</tbody>
</table>
If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>function</th>
<th>The name of the function with which to associate the comment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>function-args</td>
<td>The function arguments.</td>
</tr>
<tr>
<td>comment</td>
<td>Specifies the comment text to add. If a comment already exists for this function, this overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. Enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td>NULL</td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>

**Privileges**

- **Superuser**: View and add comments to all objects.
- **Object owner**: Add or edit comments for the object.
- **User**: VIEW privileges on an object to view its comments.

**Examples**

The following example adds a comment to the `macros.zerowhennull (x INT)` function:

```sql
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS 'Returns a 0 if not NULL';
```

The following examples remove a comment from the `macros.zerowhennull (x INT)` function in two ways by using the NULL option, or specifying a blank string:

```sql
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS NULL;
=> COMMENT ON FUNCTION macros.zerowhennull(x INT) IS '';
```

**COMMENT ON LIBRARY**

Adds, revises, or removes a comment on a library. Each object can have one comment. Comments are stored in the system table `COMMENTS`. 
Syntax

```
COMMENT ON LIBRARY [[database.]schema.]library IS {'comment' | NULL}
```

Parameters

| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:
|                   | myschema.thisDbObject |
| library           | The name of the library associated with the comment. |
| comment            | Specifies the comment text to add. If a comment already exists for this library, this comment overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. Enclose a blank value within single quotes to remove an existing comment. |
| NULL               | Removes an existing comment. |

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the library MyFunctions:
The following examples remove a comment from the library MyFunctions:

```
=> COMMENT ON LIBRARY MyFunctions IS NULL;
=> COMMENT ON LIBRARY MyFunctions IS '';
```

See Also

- COMMENTS

**COMMENT ON NODE**

Adds, revises, or removes a comment on a node. Each object can have one comment. Comments are stored in the system table COMMENTS.

Dropping an object drops all comments associated with the object.

**Syntax**

```
COMMENT ON NODE node-name IS { 'comment' | NULL }
```

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>node-name</code></td>
<td>The name of the node associated with the comment.</td>
</tr>
<tr>
<td><code>comment</code></td>
<td>Specifies the comment text to add. If a comment already exists for this node, this comment overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. Enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td><code>NULL</code></td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>
Privileges

- **Superuser**: View and add comments to all objects.
- **Object owner**: Add or edit comments for the object.
- **User**: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment for the `initiator` node:

```sql
=> COMMENT ON NODE initiator IS 'Initiator node';
```

The following examples removes a comment from the `initiator` node.

```sql
=> COMMENT ON NODE initiator IS NULL;
=> COMMENT ON NODE initiator IS '';
```

See Also

`COMMENTS`

**COMMENT ON PROJECTION**

Adds, revises, or removes a comment on a projection. Each object can have one comment. Comments are stored in the system table `COMMENTS`.

Dropping an object drops all comments associated with the object.

Syntax

```sql
COMMENT ON PROJECTION [[database.]schema.]projection IS { 'comment' | NULL }
```

Parameters

|   | Specifies a schema, by default public. If `schema` is any schema |
For example:

```
myschema.thisDbObject
```

If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th><strong>database.[schema</strong></th>
<th>other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projection</strong></td>
<td>The name of the projection associated with the comment.</td>
</tr>
<tr>
<td><strong>comment</strong></td>
<td>Specifies the text of the comment to add. If a comment already exists for this projection, the comment you enter here overwrites the previous comment. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. Enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td><strong>NULL</strong></td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>

**Privileges**

- **Superuser:** View and add comments to all objects.
- **Object owner:** Add or edit comments for the object.
- **User:** VIEW privileges on an object to view its comments.

**Examples**

The following example adds a comment to the `customer_dimension_vmart_node01` projection:

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS 'Test data';
```

The following examples remove a comment from the `customer_dimension_vmart_node01` projection:

```
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS NULL;
=> COMMENT ON PROJECTION customer_dimension_vmart_node01 IS '';
```
See Also

COMMENTS

COMMENT ON SCHEMA

Adds, revises, or removes a comment on a schema. Each object can have one comment. Comments are stored in the system table COMMENTS.

Syntax

COMMENT ON SCHEMA schema-name IS ('comment' | NULL)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema-name</td>
<td>The schema associated with the comment.</td>
</tr>
<tr>
<td>comment</td>
<td>Text of the comment to add. If a comment already exists for this schema, the comment you enter here overwrites the previous comment. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. You can enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td>NULL</td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.
Examples

The following example adds a comment to the public schema:

```sql
=> COMMENT ON SCHEMA public IS 'All users can access this schema';
```

The following examples remove a comment from the public schema.

```sql
=> COMMENT ON SCHEMA public IS NULL;
=> COMMENT ON SCHEMA public IS '';
```

**COMMENT ON SEQUENCE**

Adds, revises, or removes a comment on a sequence. Each object can have one comment. Comments are stored in the system table `COMMENTS`.

**Syntax**

```
COMMENT ON SEQUENCE [[database.]schema.]sequence IS { 'comment' | NULL }
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td><code>sequence</code></td>
<td>The name of the sequence associated with the comment.</td>
</tr>
</tbody>
</table>
| `comment` | Specifies the text of the comment to add. If a comment already exists for this sequence, this comment overwrites the previous one. Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message. You can enclose a blank value within single quotes to remove an
existing comment.
NULL    Removes an existing comment.

Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the sequence called prom_seq.

```=> COMMENT ON SEQUENCE prom_seq IS 'Promotion codes';```

The following examples remove a comment from the prom_seq sequence.

```=> COMMENT ON SEQUENCE prom_seq IS NULL;
=> COMMENT ON SEQUENCE prom_seq IS '';```

COMMENT ON TABLE

Adds, revises, or removes a comment on a table. Each object can have one comment. Comments are stored in the system table COMMENTS.

Syntax

```COMMENT ON TABLE [[database.]schema.]table IS { 'comment' | NULL }```

Parameters

```
[ database.]schema Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:
```
If you specify a database, it must be the current database.

**table**
The name of the table with which to associate the comment.

**comment**
Specifies the text of the comment to add. Enclose the text of the comment within single-quotes. If a comment already exists for this table, the comment you enter here overwrites the previous comment.

Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.

You can enclose a blank value within single quotes to remove an existing comment.

**NULL**
Removes a previously added comment.

**Privileges**

- **Superuser**: View and add comments to all objects.
- **Object owner**: Add or edit comments for the object.
- **User**: VIEW privileges on an object to view its comments

**Examples**

The following example adds a comment to the promotion_dimension table:

```sql
=> COMMENT ON TABLE promotion_dimension IS '2011 Promotions';
```

The following examples remove a comment from the promotion_dimension table:

```sql
=> COMMENT ON TABLE promotion_dimension IS NULL;
=> COMMENT ON TABLE promotion_dimension IS '';
```
COMMENT ON TRANSFORM FUNCTION

Adds, revises, or removes a comment on a user-defined transform function. Each object can have one comment. Comments are stored in the system table COMMENTS.

Syntax

```
COMMENT ON TRANSFORM FUNCTION [[database.]schema.]tfunction
...( [ tfunction-arg-name tfunction-arg-type ][,...] ) IS ('comment' | NULL)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>tfunction</code></td>
<td>The name of the transform function with which to associate the comment.</td>
</tr>
<tr>
<td><code>tfunction-arg-name</code></td>
<td>The names and data types of one or more transform function arguments. If you supply argument names and types, each type must match the type specified in the library used to create the original transform function.</td>
</tr>
<tr>
<td><code>tfunction-arg-type</code></td>
<td></td>
</tr>
<tr>
<td><code>comment</code></td>
<td>Specifies the comment text to add. If a comment already exists for this transform function, this comment overwrites the previous one.</td>
</tr>
<tr>
<td></td>
<td>Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.</td>
</tr>
<tr>
<td></td>
<td>Enclose a blank value within single quotes to remove an existing comment.</td>
</tr>
<tr>
<td><code>NULL</code></td>
<td>Removes an existing comment.</td>
</tr>
</tbody>
</table>
Privileges

- Superuser: View and add comments to all objects.
- Object owner: Add or edit comments for the object.
- User: VIEW privileges on an object to view its comments.

Examples

The following example adds a comment to the `macros.zerowhennull (x INT) UTF` function:

```sql
=> COMMENT ON TRANSFORM FUNCTION macros.zerowhennull(x INT) IS 'Returns a 0 if not NULL';
```

The following example removes a comment from the `acros.zerowhennull (x INT)` function by using the NULL option:

```sql
=> COMMENT ON TRANSFORM FUNCTION macros.zerowhennull(x INT) IS NULL;
```

COMMENT ON VIEW

Adds, revises, or removes a comment on a view. Each object can have one comment. Comments are stored in the system table `COMMENTS`.

Syntax

```
COMMENT ON VIEW [[database.]schema.]view IS { 'comment' | NULL }
```

Parameters

```
[ database.]schema

Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:

```
myschema.thisDbObject
```

If you specify a database, it must be the current database.
```
**view**  | The name of the view with which to associate the comment.
---|---
**comment**  | Specifies the text of the comment to add. If a comment already exists for this view, this comment overwrites the previous one.
  Comments can be up to 8192 characters in length. If a comment exceeds that limitation, Vertica truncates the comment and alerts the user with a message.
  Enclose a blank value in single quotes to remove an existing comment.
**NULL**  | Removes an existing comment.

### Privileges

- **Superuser**: View and add comments to all objects.
- **Object owner**: Add or edit comments for the object.
- **User**: VIEW privileges on an object to view its comments.

### Examples

The following example adds a comment to a view called `curr_month_ship`:

```sql
=> COMMENT ON VIEW curr_month_ship IS 'Shipping data for the current month';
```

The following example removes a comment from the `curr_month_ship` view:

```sql
=> COMMENT ON VIEW curr_month_ship IS NULL;
```

### COMMIT

Ends the current transaction and makes all changes that occurred during the transaction permanent and visible to other users.

COMMIT is a synonym for **END**
Syntax

`COMMIT [ WORK | TRANSACTION ]`

Parameters

| WORK   | TRANSACTION   | Optional keywords for readability only. |

Privileges

None

Examples

This example shows how to commit an insert.

```sql
=> CREATE TABLE sample_table (a INT);
=> INSERT INTO sample_table (a) VALUES (1);
OUTPUT
-------
1
=> COMMIT;
```

See Also

- Transactions
- Creating Transactions
- BEGIN
- ROLLBACK
- START TRANSACTION
CONNECT

Connects to another Vertica database to enable data import (using the COPY FROM VERTICA statement) or export (using the EXPORT statement). By default, invoking CONNECT occurs over the Vertica private network. Creating a connection over a public network requires some configuration. For information about using CONNECT to export data to or import data over a public network, see Using Public and Private IP Networks.

When importing from or exporting to a Vertica database, you can connect only to a database that uses trusted- (username-only) or password-based authentication, as described in Security and Authentication. SSL authentication is not supported.

Syntax

CONNECT TO VERTICA database USER username PASSWORD 'password' ON 'host',port

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>The connection target database name.</td>
</tr>
<tr>
<td>username</td>
<td>The username to use when connecting to the other database.</td>
</tr>
<tr>
<td>password</td>
<td>A string containing the password to use to connect to the other database.</td>
</tr>
<tr>
<td>host</td>
<td>A string containing the host name of one of the nodes in the other database.</td>
</tr>
<tr>
<td>port</td>
<td>The port number of the other database as an integer.</td>
</tr>
</tbody>
</table>

Privileges

None

Connection Details

After you establish a connection to another database, the connection remains open for the current session. To disconnect a connection, use the DISCONNECT statement.
You can have only one connection to another database at a time. However, you can create connections to multiple different databases in the same session.

If the target database does not have a password, and you specify a password in the CONNECT statement, the connection succeeds, but does not give any indication that you supplied an incorrect password.

**Example**

```sql
=> CONNECT TO VERTICA ExampleDB USER dbadmin PASSWORD 'Password123' ON 'VerticaHost01',5433;
CONNECT
```

**See Also**

- COPY FROM VERTICA
- EXPORT TO VERTICA

**COPY**

COPY bulk-loads data into a Vertica database. By default, COPY automatically commits itself and any current transaction except when loading temporary tables. If COPY is terminated or interrupted Vertica rolls it back.

COPY reads data as UTF-8 encoding.

For information on loading one or more files or pipes on a cluster host or on a client system, see COPY LOCAL.

**Syntax**

```sql
COPY [[schema-name.]]target-table
  ...
  [ ( { column-as-expression | column } ]
  ....... [ DELEMITER [ AS ] 'char' ]
  ....... [ ENCLOSED [ BY ] 'char' ]
  ....... [ ENFORCELENGTH ]
  ....... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
  ....... [ FILLER datatype]
  ....... [ FORMAT 'format' ]
  ....... [ NULL [ AS ] 'string' ]
  ....... [ TRIM 'byte' ]
  ...
) ]
```
... [ COLUMN OPTION ( column
...... [ DELIMITER [ AS ] 'char' ]
...... [ ENCLOSED [ BY ] 'char' ]
...... [ ENFORCELENGTH ]
...... [ ESCAPE [ AS ] 'char' | NO ESCAPE ]
...... [ FORMAT 'format' ]
...... [ NULL [ AS ] 'string' ]
...... [ TRIM 'byte' ]
... ([,...] ) ]
 [ FROM {
STDIN [ input-format ]
[ 'path-to-data' [ ON { nodename | (nodeset) | ANY NODE } ] [ input-format ] ] [,...]
[ LOCAL {STDIN | 'path-to-data'} [ input-format ] [,...]
 [ VERTICA source-database.[source-schema.]source-table[(source-column [,...])]
} ]
...[ [ WITH ] PARSER parser ( [ arg=value[,...] ] ) ]
...[ [ WITH ] UDL-clause[,...] ]
...[ DELIMITER [ AS ] 'char' ]
...[ TRAILING NULLCOLS ]
...[ NULL [ AS ] 'string' ]
...[ ESCAPE [ AS ] 'char' | NO ESCAPE ]
...[ ENCLOSED [ BY ] 'char' ]
...[ RECORD TERMINATOR 'string' ]
...[ SKIP records ]
...[ SKIP BYTES integer ]
...[ TRIM 'byte' ]
...[ REJECTMAX integer ]
...[ REJECTED DATA ( [path' [ ON nodename ] [,...] | AS TABLE reject-table] ]
...[ EXCEPTIONS 'path' [ ON nodename ] [,...] ]
...[ ENFORCELENGTH ]
...[ ERROR TOLERANCE ]
...[ ABORT ON ERROR ]
...[ [ STORAGE ] load-method ]
...[ STREAM NAME 'streamName']
...[ NO COMMIT ]

Parameters
See COPY Parameters

Privileges

General

Superusers have full COPY privileges. The following requirements apply to non-superusers:

- USER-accessible storage location

- Applicable READ or WRITE privileges granted to the storage location where files are read or written
COPY LOCAL

- INSERT privileges to copy data from the STDIN pipe
- USAGE privileges on the schema

COPY FROM STDIN

- INSERT privilege on table
- USAGE privilege on schema

File Paths for Rejected Data and Exceptions

COPY can specify a path to store rejected data and exceptions. If the path resolves to a storage location, the following privileges apply to non-superusers:

- The storage location was created with the USER option (see CREATE LOCATION).
- The user must have READ access to the storage location, as described in GRANT (Storage Location)

COPY Topics

COPY Option Parser Dependencies
COPY Restrictions
Setting vsql Variables
See Also

COPY Parameters

Note: For details on which COPY parameters are valid for specific parsers, see COPY Option Parser Dependencies.

| database | schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject |
If you specify a database, it must be the current database. COPY ignores `schema-name` when used in `CREATE EXTERNAL TABLE` or `CREATE FLEX EXTERNAL TABLE` statements.

<table>
<thead>
<tr>
<th><strong>target-table</strong></th>
<th>The target columnar or flexible table for loading new data. Vertica loads the data into all projections that include columns from the schema table.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>column-as-expression</strong></td>
<td>Specifies the expression used to compute values for the target column. For example: <code>COPY t(year AS TO_CHAR(k, 'YYY')) FROM 'myfile.dat'</code> Use this option to transform data when it is loaded into the target database. For details about:</td>
</tr>
<tr>
<td><strong>column</strong></td>
<td>Restricts the load to one or more specified columns in the table. If you do not specify any columns, COPY loads all columns by default. Table columns that you do not specify in the column list are assigned their default values. If a column had no defined default value, COPY inserts NULL. If you leave the column parameter blank to load all columns in the table, you can use the optional parameter <code>COLUMN OPTION</code> to specify parsing options for specific columns. The data file must contain the same number of columns as the COPY command's column list. For example, in a table T1 with nine columns (C1 through C9), the following statement loads the three columns of data in each record to columns C1, C6, and C9, respectively: <code>COPY T1 (C1, C6, C9);</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### FILLER `datatype`

Specifies not to load the column and its fields into the destination table. Use this option to omit columns that you do not want to transfer into a table.

This parameter also transforms data from a source column and loads the transformed data to the destination table, rather than loading the original, untransformed source column (parsed column).

**Note:** If `datatype` is VARCHAR, set the VARCHAR length (VARCHAR(`n`)) so the combined length of all FILLER source fields does not exceed the target column's defined length. Otherwise, the COPY command can return with an error.

For more information, see Manipulating Source Data Columns in the Administrator's Guide.

### FORMAT `'format'`

Specifies the input formats to use when loading `date/time` and `binary` columns, where `format` can be one of the following:

These are the valid input formats when loading binary columns:

- octal
- hex
- bitstream

See Loading Binary (Native) Data to learn more about these formats.

When loading `date/time` columns, using FORMAT significantly improves load performance. COPY supports the same formats as the TO_DATE function.

See the following topics for additional information:

- Template Patterns for Date/Time Formatting
- Template Pattern Modifiers for Date/Time Formatting

If you specify invalid format strings, the COPY operation returns an error.
**path-to-data**

Specifies the absolute path of the file (or files) containing the data, which can be from multiple input sources. If the file is stored in HDFS, *path-to-data* is a URL in the hdfs scheme, typically hdfs:///path/to/file. See [Reading Directly from HDFS](#).

For Parquet, ORC, text, and delimited files only, *path-to-data* can be the URL of an S3 bucket in the format 's3://bucket/path'.

*path-to-data* can optionally contain wildcards to match more than one file. The file or files must be accessible to the local client or the host on which the COPY statement runs. COPY skips empty files in the file list. A file list that includes directories causes the query to fail. See [Specifying COPY FROM Options](#).

The supported patterns for wildcards are specified in the [Linux Manual Page for Glob (7)](#), and for ADO.net platforms, through the .NET [Directory.GetFiles Method](#).

You can use variables to construct the pathname as described in [Using Load Scripts](#).

If *path* resolves to a storage location on a local file system, and the user invoking COPY is not a superuser, the following privileges apply:

- The storage location must have been created with the USER option (see [CREATE LOCATION](#)).

- The user must already have been granted READ access to the file storage location, as described in [GRANT (Storage Location)](#).

Further, if a user has privileges but is not a superuser, and invokes COPY from that storage location, Vertica ensures that symbolic links do not result in unauthorized access.

**ON nodename**

Specifies the node on which the data to copy resides and the node that should parse the load file. If you omit *nodename*, the location of the input file defaults to the COPY initiator node. Use *nodename* to copy and parse a load file from a node other than the COPY initiator node.

**Note:** *nodename* is invalid with STDIN and LOCAL: STDIN
<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
</table>
| **ON** **(`nodeset`)** | Specifies a set of nodes on which to perform the load. The same data must be available for load on all named nodes. `nodeset` is a comma-separated list of node names in parentheses. For example:  
   
   ```sql
   => COPY t FROM 'file1.txt' ON (v_vmart_node0001, v_vmart_node0002);
   ```  
   Vertica apportions the load among all of the specified nodes. If you also specify `ERROR TOLERANCE` or `REJECTMAX`, Vertica instead chooses a single node on which to perform the load.  
   
   If the data is available on all nodes, you usually use `ON ANY NODE`. However, you can use `nodeset` to do manual load-balancing among concurrent loads. |
| **ON ANY NODE** | Specifies that the source file to load is available on all of the nodes, so `COPY` opens the file and parses it from any node(s) in the cluster.  
   
   **Caution:** The file must be the same on all nodes. If the file differs on two nodes, an incorrect or incomplete result is returned, with no error or warning.  
   
   Vertica attempts to apportion the load among several nodes if the file is large enough to benefit from apportioning. It chooses a single node if `ERROR TOLERANCE` or `REJECTMAX` is specified.  
   
   You can use a wildcard or glob (such as `*.dat`) to load multiple input files, combined with the `ON ANY NODE` clause. If you use a glob, `COPY` distributes the list of files to all cluster nodes and spreads the workload.  
   
   `ON ANY NODE` is invalid with `STDIN` and `LOCAL`: `STDIN` can only use the initiator node, and `LOCAL` indicates a client node.  
   
   `ON ANY NODE` is the default for HDFS and S3 paths and does not need to be specified. |
| **STDIN**     | Reads from the client a standard input instead of a file. `STDIN` is read on the initiator node only, and `LOCAL` indicates a client node. |
takes one input source only and is read on the initiator node. To load multiple input sources, use path-to-data.

User must have INSERT privileges on the table and USAGE privileges on its schema.

| LOCAL | \{STDIN | 'path-to-data'\} |
|-------|--------------------------|
|       | Specifies that all paths for the COPY statement are on the client system and that all COPY variants are initiated from a client. You can use LOCAL and path-to-data (see above), to specify a relative path. For details, see COPY LOCAL. |

<table>
<thead>
<tr>
<th>input-format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the input format, one of the following:</td>
</tr>
<tr>
<td>- UNCOMPRESSED (default)</td>
</tr>
<tr>
<td>- BZIP</td>
</tr>
<tr>
<td>- GZIP</td>
</tr>
<tr>
<td>- LZO</td>
</tr>
</tbody>
</table>

Input files can be of any format. If you use wildcards, all qualifying input files must be in the same format. To load different file formats, specify the format types specifically.

The following requirements and restrictions apply:

- When using concatenated BZIP or GZIP files, verify that all source files terminate with a record terminator before concatenating them.

- Concatenated BZIP and GZIP files are not supported for NATIVE (binary) and NATIVE VARCHAR formats.

- LZO files are assumed to be compressed with lzop. Vertica supports the following lzop arguments:
  --no-checksum / -F
  --crc32
  --adler32
  --no-name / -n
  --name / -N
  --no-mode
  --no-time
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression Levels</td>
<td>--fast, --best, Numbered compression levels</td>
</tr>
<tr>
<td></td>
<td>- BZIP, GZIP, and LZO compression cannot be used with ORC format.</td>
</tr>
<tr>
<td>Parser</td>
<td>Specifies the parser to use when bulk loading columnar tables, one of the following:</td>
</tr>
<tr>
<td></td>
<td>- NATIVE</td>
</tr>
<tr>
<td></td>
<td>- NATIVE VARCHAR</td>
</tr>
<tr>
<td></td>
<td>- FIXEDWIDTH</td>
</tr>
<tr>
<td></td>
<td>- ORC[(hive_partition_cols='partitions')]</td>
</tr>
<tr>
<td></td>
<td>- PARQUET[(hive_partition_cols='partitions')]</td>
</tr>
</tbody>
</table>

By default, COPY uses the DELIMITER parser for UTF-8 format, delimited text input data.

To use a flex table parser for column tables, use the PARSER parameter followed by a flex table parser argument. For parser descriptions, see Flex Parsers Reference.

**Note:** You do not specify the DELIMITER parser directly; absence of a specific parser indicates the default.

For more information, see Specifying a COPY Parser in the Administrator's Guide.

The following restrictions apply:

- These parsers are not applicable when loading flexible tables.
- The ORC and PARQUET parsers are for use with Hadoop files in those formats. The files do not need to be stored in HDFS. For details, see Reading Hadoop Columnar File Formats.
- COPY LOCAL does not support NATIVE and NATIVE VARCHAR parsers.
<table>
<thead>
<tr>
<th><strong>[WITH]</strong> <strong>UDL-clause[...]</strong></th>
<th>Specifies one or more user-defined load functions—one source, one or more filters, and one parser, as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>SOURCE</strong> source( [arg=value[,..] ] )</td>
<td></td>
</tr>
<tr>
<td>• <strong>FILTER</strong> filter( [arg=value[,..] ] )</td>
<td></td>
</tr>
<tr>
<td>• <strong>PARSER</strong> parser( [arg=value[,..] ] )</td>
<td></td>
</tr>
</tbody>
</table>

To use a flex table parser for column tables, use the PARSER parameter followed by a flex table parser argument. For supported flex table parsers, see [Bulk Loading Data into Flex Tables](#).

<table>
<thead>
<tr>
<th><strong>COLUMN OPTION</strong></th>
<th>Specifies load metadata for one or more columns declared in the table column list. For example, you can specify that a column has its own DELIMITER, ENCLOSED BY, NULL as 'NULL' expression, and so on. You do not have to specify every column name explicitly in the COLUMN OPTION list, but each column you specify must correspond to a column in the table column list.</th>
</tr>
</thead>
</table>

| **COLSIZES** (**integer [,...]**)) | Specifies column widths when loading fixed-width data. COPY requires that you specify the COLSIZES when using the FIXEDWIDTH parser. COLSIZES and the list of integers must correspond to the columns listed in the table column list. For more information, see [Loading Fixed-Width Format Data](#) in the Administrator's Guide. |

| **DELMITER** | Indicates the single ASCII character used to separate columns within each record of a file. You can use any ASCII value in the range E'\000' to E'\177', inclusive. You cannot use the same character for both the DELIMITER and NULL parameters. For more information, see [Loading Delimited Data](#) in the Administrator's Guide. |

**Default:** Vertical bar (|).
| **TRAILING NULLCOLS** | Specifies that if Vertica encounters a record with insufficient data to match the columns in the table column list, COPY inserts the missing columns with NULLs. For other information and examples, see [Loading Fixed-Width Format Data](#) in the Administrator's Guide.  
This option is not available for ORC or Parquet data. |
| **ESCAPE [AS]** | Sets the escape character. Once set, the character following the escape character is interpreted literally, rather than as a special character. You can define an escape character using any ASCII value in the range `E'\001'` to `E'\177'`, inclusive (any ASCII character except NULL: `E'\000'`). Note that the backslash character (`\`) is the default escape character.  
The COPY statement does not interpret the data it reads in as [String Literals](#). It also does not follow the same escape rules as other SQL statements (including the COPY parameters). When reading in data, COPY interprets only the characters defined by these options as special values:  
- ESCAPE [AS]  
- DELIMITER  
- ENCLOSED [BY]  
- RECORD TERMINATOR |
| **NO ESCAPE** | Eliminates escape character handling. Use this option if you do not need any escape character and you want to prevent characters in your data from being interpreted as escape sequences. |
| **ENCLOSED [BY]** | Sets the quote character within which to enclose data, allowing delimiter characters to be embedded in string values. You can choose any ASCII value in the range `E'\001'` to `E'\177'` inclusive (any ASCII character except NULL: `E'\000'`). By default, ENCLOSED BY has no value, meaning data is not enclosed by any sort of quote character. |
| **NULL** | The string representing a null value. The default is an empty string (`''`). You can specify a null value as any ASCII value in the range `E'\001'` to `E'\177'` inclusive (any ASCII character
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD TERMINATOR</td>
<td>Specifies the literal character string indicating the end of a data file record. For more information about using this parameter, see Loading Delimited Data. This option is not available for ORC or Parquet data.</td>
</tr>
<tr>
<td>SKIP records</td>
<td>Indicates the number (integer) of records to skip in a load file. For example, you can use the SKIP option to omit table header information. This option is not available for ORC or Parquet data.</td>
</tr>
<tr>
<td>SKIP BYTES total</td>
<td>Indicates the total number (integer) of bytes in a record to skip. This option is only available when loading fixed-width data. This option is not available for ORC or Parquet data.</td>
</tr>
<tr>
<td>TRIM</td>
<td>Trims the number of bytes you specify from a column. This option is only available when loading fixed-width data. You can set TRIM at the table level for a column, or as part of the COLUMN OPTION parameter. This option is not available for ORC or Parquet data.</td>
</tr>
<tr>
<td>REJECTMAX</td>
<td>Specifies a maximum number of logical records that can be rejected before a load fails. For details, see Capturing Load Rejections and Exceptions. REJECTMAX disables apportioned load.</td>
</tr>
<tr>
<td>REJECTED DATA</td>
<td>Specifies where to write each row that failed to load. If this parameter is specified, records that failed due to parsing errors are always written. Records that failed due to an error during a transformation are written only if the CopyFaultTolerantExpressions configuration parameter is set. (See General Parameters.) The syntax for this parameter is:</td>
</tr>
</tbody>
</table>

```sql
REJECTED_DATA
{ 'path' [ ON nodename ] [,..] | AS TABLE reject-table }
```

Vertica can write rejected data to the specified path or to a
### table:

- `'path' [ ON nodename ]`: Copies the rejected row data to the specified path on the node executing the load. If qualified by ON `nodename`, Vertica moves existing rejected data files on `nodename` to `path` on the same node.

The value of `path` can be a directory or a file prefix. If there are multiple load sources, `path` is always treated as a directory. If there are not multiple load sources but `path` ends with `/`, or if a directory of that name already exists, it is also treated as a directory. Otherwise, `path` is treated as a file prefix.

- `REJECTED DATA AS TABLE reject-table`: Saves rejected rows to columnar table `reject-table`.

When this parameter is used with COPY...ON ANY NODE, rejected data processed by each node is written to `path` on that node. To collect all rejected data in one place regardless of how the load is distributed, use a table.

For details about both options, see Capturing Load Rejections and Exceptions in the Administrator's Guide.

### EXCEPTIONS

Specifies the file name or absolute path of the file in which to write exceptions, as follows:

```
EXCEPTIONS 'path' [ ON nodename[, ...] ]
```

Files are written on the node or nodes executing the load.

Exceptions describe why each rejected row was rejected. Each exception describes the corresponding record in the file specified by the REJECTED DATA option.

If `path` resolves to a storage location, the following privileges apply to non-superusers:

- The storage location must be created with the USER option (see CREATE LOCATION).

- The user must have READ access to the storage location where the files exist, as described in GRANT (Storage Location).
The **ON nodename** clause moves existing exceptions files on `nodename` to the indicated **path** on the same node. For details, see Saving Load Exceptions (EXCEPTIONS) in the Administrator's Guide.

When this parameter is used with **COPY...ON ANY NODE**, exceptions processed by each node are written to **path** on that node.

Specifying an exceptions file name is incompatible with the **REJECTED DATA AS TABLE** clause. Exceptions are listed in the table's rejected_reason column.

| **ENFORCELENGTH** | Determines whether **COPY** truncates or rejects data rows of type char, varchar, binary, and varbinary if they do not fit the target table. Specifying the optional ENFORCELENGTH parameter rejects rows.  
You can set ENFORCELENGTH at the table level for a column, or as part of the COLUMN OPTION parameter.  
**Default**: **COPY** truncates offending rows of these data types, but does not reject them. For more details, see Tracking Load Exceptions and Rejections Status in the Administrator's Guide. |
|-------------------|---------------------------------------------------------------------------------------------------------------|
| **ERROR TOLERANCE** | Specifies that **COPY** treats each source during execution independently when loading data. The statement is not rolled back if a single source is invalid. The invalid source is skipped and the load continues.  
This option is not available for ORC or Parquet data.  
Using this parameter disables apportioned load. |
| **ABORT ON ERROR** | Specifies that **COPY** stops if any row is rejected. The statement is rolled back and no data is loaded. |

**[STORAGE]**

**Load-method**

Specifies how to load data into the database, one of the following:

- **AUTO** (default): Initially loads data into WOS, suitable for smaller bulk loads.

- **DIRECT**: Loads data directly into ROS containers, suitable for large (>100 MB) bulk loads.
TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.

This option is invalid for external tables.

For details, see Choosing a Load Method in the Administrator's Guide.

| STREAM NAME | [Optional] Supplies a COPY load stream identifier. Using a stream name helps to quickly identify a particular load. The STREAM_NAME value that you supply in the load statement appears in the stream column of the LOAD_STREAMS system table.

A valid stream name can contain any combination of alphanumeric or special characters up to 128 bytes in length.

By default, Vertica names streams by table and file name. For example, if you are loading two files (f1, f2) into TableA, their default stream names are TableA-f1, TableA-f2, respectively.

To name a stream:
=> COPY mytable FROM myfile DELIMITER '|' DIRECT STREAM_NAME 'My stream name'; |

| NO COMMIT | Prevents the COPY statement from committing its transaction automatically when it finishes copying data.

The following requirements and restrictions apply:

- This option must be the last COPY statement parameter.

- CREATE EXTERNAL TABLE AS COPY ignores this option.

For more information, see Overriding COPY Auto Commit.

This option is not available for ORC or Parquet data.

COPY Option Parser Dependencies

The following table summarizes which COPY parameters are available when loading data using the default (DELIMITER), NATIVE (binary), NATIVE VARCHAR, and FIXEDWIDTH parsers:
<table>
<thead>
<tr>
<th>COPY Option</th>
<th>DELIMITER</th>
<th>NATIVE (BINARY)</th>
<th>NATIVE (VARCHAR)</th>
<th>FIXEDWIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN OPTION</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>AUTO</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>DIRECT</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>TRICKLE</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>ENFORCELENGTH</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>EXCEPTIONS</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>FILLER</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>REJECTED DATA</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>ABORT ON ERROR</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>STREAM NAME</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SKIP</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SKIP BYTES</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REJECTMAX</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>STDIN</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>UNCOMPRESSED</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>BZIP</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>GZIP</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>CONCATENATED BZIP GZIP</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO COMMIT</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>FORMAT</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>NULL</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>DELIMITED</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENCLOSED BY</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**COPY Restrictions**

### Invalid Data

COPY considers the following data invalid:

- **Missing columns** (an input line has fewer columns than the recipient table).
- **Extra columns** (an input line has more columns than the recipient table).
- **Empty columns** for an INTEGER or DATE/TIME data type. If a column is empty for either of these types, COPY does not use the default value that was defined by `CREATE TABLE`. However, if you do not supply a column option as part of the COPY statement, the default value is used.
- **Incorrect representation** of a data type. For example, trying to load a non-numeric value into an INTEGER column is invalid.

### Constraint Violations

If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements.

**Note:** Automatic constraint enforcement requires that you have SELECT privileges on the table containing the constraint.
Empty Line Handling

When COPY encounters an empty line while loading data, the line is neither inserted nor rejected, but COPY increments the line record number. Consider this behavior when evaluating rejected records. If you return a list of rejected records and COPY encountered an empty row while loading data, the position of rejected records is incremented by one.

Compressed File Errors

When loading compressed files, COPY might abort and report an error, if the file seems to be corrupted. For example, this behavior can occur if reading the header block fails.

COPY Examples

The following examples show how to load data with the COPY statement using various string options.

The FORMAT, DELIMITER, NULL, and ENCLOSED BY options:

```sql
=> COPY public.customer_dimension (customer_since FORMAT 'YYYY')
   FROM STDIN
   DELIMITER ','
   NULL AS 'null'
   ENCLOSED BY '"';
```

The DELIMITER, NULL, and DIRECT options:

```sql
=> COPY a
   FROM STDIN
   DELIMITER '__,__'
   NULL E'\N'
   DIRECT;
```

The DELIMITER and NULL options:

```sql
=> COPY store.store_dimension
   FROM :input_file
   DELIMITER '|' 
   NULL ''
   RECORD TERMINATOR E'\f';
```
Setting vsql Variables

The first two examples load data from STDIN. The last example uses a vsql variable (`input_file`). You can set a vsql variable as follows:

```sql
=> \set input_file ../myCopyFromLocal/large_table.gzip
```

Including Multiple Source Files

COPY supports the inclusion of multiple source files in a single COPY statement.

The following example creates a table named 'sampletab.' It then copies multiple source files to the table using a single COPY statement.

```sql
=> CREATE TABLE sampletab (a int);
CREATE TABLE
=> COPY sampletab FROM '/home/dbadmin/one.dat', 'home/dbadmin/two.dat';
Rows Loaded
--------------
 2
(1 row)
```

You can use wildcards to indicate a group of files:

```sql
=> COPY myTable FROM 'hdfs:///mydirectory/ofmanyfiles/*.dat';
```

Wildcards can include regular expressions:

```sql
=> COPY myTable FROM 'hdfs:///mydirectory/*_[0-9]';
```

You can use multiple paths in a single COPY statement:

```sql
=> COPY myTable FROM 'hdfs:///data/sales/01/*.dat', 'hdfs:///data/sales/02/*.dat', 'hdfs:///data/sales/historical.dat';
```

Distributing a Load

The following example shows how you can load data that is shared across all nodes. Vertica distributes the load across all nodes, if possible.

```sql
=> COPY sampletab FROM '/data/file.dat' ON ANY NODE;
```
This example shows how to load data from two files. Because the first load file does not specify nodes (or ON ANY NODE), the initiator performs the load. Loading the second file is distributed across all nodes.

```
=> COPY samplertab FROM '/data/file1.dat', '/data/file2.dat' ON ANY NODE;
```

This example shows how to specify different nodes for each load file. Vertica distributes the load of file1.dat across v_vmart_node0001 and v_vmart_node0002 and distributes the load of file2.dat across v_vmart_node0003 and v_vmart_node0004.

```
=> COPY samplertab FROM '/data/file1.dat' ON (v_vmart_node0001, v_vmart_node0002),
    '/data/file2.dat' ON (v_vmart_node0003, v_vmart_node0004);
```

ON ANY NODE is the default for loads from HDFS. You do not need to specify it.

### Loading Data from Shared Storage

To load data from HDFS or S3, use URLs in the corresponding schemes—hdfs:///path or s3://bucket/path. Loads from HDFS and S3 default to ON ANY NODE; you do not need to specify it.

This example shows how you can load a file stored in HDFS using the default name node or name service. See [Reading Directly from HDFS](#) for more information about HDFS URLs.

```
=> COPY t FROM 'hdfs:///opt/data/file1.dat';
```

This example shows how you can load data from a particular HDFS name service (testNS). You specify a name service if your database is configured to read from more than one HDFS cluster.

```
=> COPY t FROM 'hdfs://testNS/opt/data/file2.csv';
```

This example shows how you can load data from an S3 bucket. See [Loading from an S3 Bucket](#) for more information.

```
=> COPY t FROM 's3://AWS_DataLake/*' ORC;
```

### Loading Hadoop Native Formats

This example shows how you can load data in the ORC format from HDFS.

```
=> COPY t FROM 'hdfs:///opt/data/sales.orc' ORC;
```

This example shows how you can load Parquet data from an S3 bucket.
Loading Data into a Flex Table

This statement creates a Flex table, and copies JSON data into the table, using the flex table parser, \texttt{fjsonparser}:

\begin{verbatim}
=> CREATE FLEX TABLE darkdata();
CREATE TABLE
=> COPY tweets FROM '/myTest/Flexible/DATA/tweets_12.json' parser fjsonparser(); Rows Loaded
------------------
  12
(1 row)
\end{verbatim}

Using Named Pipes

COPY supports named pipes that follow the same naming conventions as file names on the given file system. Permissions are open, write, and close.

This statement creates the named pipe, \texttt{pipe1}, and sets two vsql variables, \texttt{dir} and \texttt{file}:

\begin{verbatim}
=> \! mkfifo pipe1
=> \set dir `pwd`/
=> \set file `''dir''pipe1''`
\end{verbatim}

This statement copies an uncompressed file from the named pipe:

\begin{verbatim}
=> \! cat pf1.dat > pipe1 &
=> COPY large_tbl FROM :file delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
\end{verbatim}

Loading Compressed Data

This statement copies a GZIP file from a named pipe and uncompresses it:

\begin{verbatim}
=> \! gzip pf1.dat
=> \! cat pf1.dat.gz > pipe1 &
=> COPY large_tbl FROM :file ON site01 GZIP delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \! gunzip pf1.dat.gz
\end{verbatim}

This statement copies a BZIP file from a named pipe and then uncompresses it:
This statement copies an LZO file from a named pipe and then uncompresses it:

```bash
=> \! bzip2 pf1.dat
=> \! cat pf1.dat.bz2 > pipe1 &
=> COPY large_tbl FROM :file ON site01 BZIP delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \! bunzip2 pf1.dat.bz2
```

```bash
=> \! lzop pf1.dat
=> \! cat pf1.dat.lzo > pipe1 &
=> COPY large_tbl FROM :file ON site01 LZO delimiter '|';
=> SELECT * FROM large_tbl;
=> COMMIT;
=> \! lzop -d pf1.dat.lzo
```

See Also

- SQL Data Types
- **ANALYZE_CONSTRAINTS**
- Choosing a Load Method in the Administrator's Guide
- **CREATE EXTERNAL TABLE AS COPY**
- Directory.getFiles Method
- Bulk-Loading Data in the Administrator's Guide
- Loading Fixed-Width Format Data in the Administrator's Guide
- Loading Binary (Native) Data in the Administrator's Guide
- Bulk Loading Data into Flex Tables in the Administrator's Guide
- Manipulating Source Data Columns in the Administrator's Guide
- Linux Manual Page for Glob (7)
- Tracking Load Exceptions and Rejections Status in the Administrator's Guide
- Transforming Data During Loads in the Administrator's Guide
COPY LOCAL

Using the COPY statement with its LOCAL option lets you load a data file on a client system, rather than on a cluster host. COPY LOCAL supports the STDIN and 'pathToData' parameters, but not the [ON nodename] clause. COPY LOCAL does not support multiple file batches in NATIVE or NATIVE VARCHAR formats. COPY LOCAL does not support reading ORC or Parquet files; use ON NODE instead. COPY LOCAL does not support CURRENT_LOAD_SOURCE().

The COPY LOCAL option is platform-independent. The statement works in the same way across all supported Vertica platforms and drivers. For more details about using COPY LOCAL with supported drivers, see the Connecting to Vertica section for your platform.

Note: On Windows clients, the path you supply for the COPY LOCAL file is limited to 216 characters due to limitations in the Windows API.

COPY LOCAL does not automatically create exceptions and rejections files, even if exceptions occur. You cannot save exceptions and rejections to a table with the rejected data as table parameter. For information about saving such files, see Capturing Load Rejections and Exceptions in the Administrator's Guide.

Privileges

User must have INSERT privilege on the table and USAGE privilege on the schema.

How Copy Local Works

COPY LOCAL loads data in a platform-neutral way. The COPY LOCAL statement loads all files from a local client system to the Vertica host, where the server processes the files. You can copy files in various formats: uncompressed, compressed, fixed-width format, in bzip or gzip format, or specified as a bash glob. Files of a single format (such as all bzip, or gzip) can be comma-separated in the list of input files. You can also use any of the applicable COPY statement options (as long as the data format supports the option). For instance, you can define a specific delimiter character, or how to handle NULLs, and so forth.

Note: The Linux glob command returns files that match the pattern you enter, as specified in the Linux Manual Page for Glob (7). For ADO.net platforms, specify patterns and wildcards as described in the .NET Directory.GetFiles Method.
For more information about using the COPY LOCAL option to load data, see COPY for syntactical descriptions, and Using COPY and COPY LOCAL for detailed examples.

The Vertica host uncompresses and processes the files as necessary, regardless of file format or the client platform from which you load the files. Once the server has the copied files, Vertica maintains performance by distributing file parsing tasks, such as encoding, compressing, uncompressing, across nodes.

**Viewing Copy Local Operations in a Query Plan**

When you use the COPY LOCAL option, the GraphViz query plan includes a label for Load-Client-File, rather than Load-File. Following is a section from a sample query plan:

```
PLAN:  BASE BULKLOAD PLAN  (GraphViz Format)

digraph G {
  graph [rankdir=BT, label = " BASE BULKLOAD PLAN \nAll Nodes Vector: \n\n node[0]=initiator (initiator) Up\n", labelloc=t, labeljust=1 ordering=out]
  .
  .
  10[label = "Load-Client-File(/tmp/diff) \nOutBlk=[UncTuple]", color = "green", shape = "ellipse"];```

**Examples**

The following example shows a load from a local file.

```
$ cat > t.dat
12
17
9
^C

=> CREATE TABLE numbers (value INT);
CREATE TABLE

=> COPY numbers FROM LOCAL 't.dat';
Rows Loaded
----------
3
(1 row)

=> SELECT * FROM numbers;
value
-----
12
17```
**COPY FROM VERTICA**

Imports data from another Vertica database. COPY FROM VERTICA is similar to COPY, but accepts only a subset of its parameters.

**Important:** The source database can be one major release behind the target database.

**Syntax**

```
COPY [[database.]schema.]target-table
... [[ target-column[,...]]]
... FROM VERTICA database.[schema.]source-table
... [[source-column[,...]]]
... [load-method]
... [STREAM NAME 'stream name']
... [NO COMMIT]
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>[database.]schema</strong></th>
<th>The table in the local database to store the copied data.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>target-column</strong></td>
<td>A target table column to store the copied data. If you specify target columns, COPY FROM VERTICA writes only to those columns. If you omit specifying target columns, Vertica writes to target table columns as described below, in Source and Target Column Mapping.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot use column fillers as part of the column definition.</td>
</tr>
<tr>
<td><strong>database</strong></td>
<td>The source database of the data to copy. A connection to this database must already exist in the current session.</td>
</tr>
<tr>
<td><strong>[schema.]source-table</strong></td>
<td>The table that is the source of the copied data.</td>
</tr>
<tr>
<td>source-column</td>
<td>A source table column to copy. If you specify source columns, only these columns are copied from the source table. If you omit specifying source columns, Vertica copies columns from the source table, as described below in Source and Target Column Mapping.</td>
</tr>
<tr>
<td>Load-method</td>
<td>Specifies how to load data into the database, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.</td>
</tr>
<tr>
<td></td>
<td>• DIRECT: Loads data directly into ROS containers, suitable for large (&gt;100 MB) bulk loads.</td>
</tr>
<tr>
<td></td>
<td>• TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.</td>
</tr>
<tr>
<td></td>
<td>This option is invalid for external tables.</td>
</tr>
<tr>
<td></td>
<td>For details, see Choosing a Load Method in the Administrator’s Guide.</td>
</tr>
<tr>
<td>STREAM NAME</td>
<td>Specifies a COPY load stream identifier. Using a stream name helps to quickly identify a particular load. The STREAM_NAME value that you specify in the load statement appears in the stream column of the LOADSTREAMS system table.</td>
</tr>
<tr>
<td>NO COMMIT</td>
<td>Prevents COPY from committing its transaction automatically when it finishes copying data. For details, see Overriding COPY Auto Commit in the Administrator's Guide.</td>
</tr>
</tbody>
</table>

### Privileges

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema
Connecting to the Source Database

Before you can import data from another database, you must establish a connection to the source database with `CONNECT`. See Copying Data from Another Vertica Database for details.

By default, `COPY FROM VERTICA` copies or imports data over the Vertica private network. Connecting to a public network requires some configuration. For information about using this statement to copy data across a public network, see Using Public and Private IP Networks.

The copy operation fails if either side of the connection is a single-node cluster installed to localhost, or you do not specify a host name or IP address.

Source and Target Column Mapping

You can optionally name a subset of source and target columns to participate in the copy operation. `COPY FROM VERTICA` attempts to match columns in the source table with corresponding columns in the destination table.

The following table compares the different combinations of naming source and target columns, and the requirements that pertain to each option.

<table>
<thead>
<tr>
<th></th>
<th>Omit source columns</th>
<th>Specify source columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Omit target columns</strong></td>
<td>Match all columns in source table to columns in target table.</td>
<td>Match named source table columns to target table columns.</td>
</tr>
<tr>
<td></td>
<td>The number of columns in the two tables can differ, but the target table must have at least as many columns as the source table.</td>
<td>The number of columns in the two tables can differ, but the target table must have at least as many columns as the number of specified source columns.</td>
</tr>
<tr>
<td><strong>Specify target columns</strong></td>
<td>Match source columns to the named target columns.</td>
<td>Match named source columns to named target columns.</td>
</tr>
<tr>
<td></td>
<td>The number of named target columns must equal the number of columns in the source table.</td>
<td>The number of source and target columns must be equal.</td>
</tr>
</tbody>
</table>
Node Failure During COPY

See Handling Node Failure During Copy/Export in the Administrator's Guide.

Examples

This example demonstrates connecting to another database, copying the contents of an entire table from the source database to an identically-defined table in the current database directly into ROS, and then closing the connection:

```sql
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD 'myPassword' ON 'VertTest01',5433;
CONNECT
=> COPY customer_dimension FROM VERTICA vmart.customer_dimension DIRECT;
Rows Loaded
----------
500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

For more examples, see Copying Data from Another Vertica Database in the Administrator's Guide.

See Also

EXPORT TO VERTICA

CREATE ACCESS POLICY

Creates a secure access policy to prevent unauthorized users from accessing potentially sensitive information. You can create access policies for table rows and columns. Access policies are a technique of on-the-fly query modification in which the query excludes rows or modifies data from a column in the results returned to the user running the query. Access policies allow different users to run the same query and receive different results.

Column access policies limit access to specific column in a table. Creating a column access policy depends on the expressions specified when creating the policy. The expression is substituted for the column's actual value in any data fetched from the table.
Row access policies limit access to a specific row in a table. You must use a WHERE clause to set the access policy's condition. Only rows that satisfy the WHERE clause are fetched from the table.

For information on how implementing access policies affects how you manage data, see Working With Access Policies in the Administrator's Guide.

Syntax

```
CREATE ACCESS POLICY ON [database.]schema.]table... { FOR COLUMN column | FOR ROWS WHERE } expression
ENABLE
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| ![database.].schema | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:  

```
myschema.thisDbObject
```

If you specify a database, it must be the current database. |
| table              | The table that contains the target column.                                  |
| column             | The column on which to apply an access policy.                              |
| expression         | An SQL expression that specifies conditions for column or row access:       |
|                    | - In a column access policy, the expression defines what value is returned when this column is fetched. The expression can contain conditions such as a role to determine what value to return. The expression might return null if no data is returned for this column. |
|                    | - In a row access policy, only rows for which the expression is true are returned. |

The expression in the example below shows the first 5 characters of the key in the `customer_key` column:  

```
=> CREATE ACCESS POLICY ON customer_dimension  
   FOR COLUMN customer_key substr(customer_key, 1, 5) ENABLE;
```

| ENABLE             | Enables the access policy. Always add this to end of the statement when creating an access policy. |
Privileges

One of the following:

- dbadmin
- Superuser

Examples

Create access policy per role on column

Add an expression to the policy to specify the access each role receives. A manager can access the complete customer number, while an operator can only see a portion of the number:

```sql
=> CREATE ACCESS POLICY on customer FOR column customer_number
    CASE
    WHEN enabled_role ('manager') then customer_number
    WHEN enabled_role ('operator') then substr (customer_number, 8,2)
    ELSE NULL
    END
    ENABLE;
```

Create access policy for rows

```sql
=> CREATE ACCESS POLICY ON customer FOR ROWS WHERE cid1>1 ENABLE;
```

See Also

In the Administrator’s Guide:

- Access Policies
- Column Access Policy
- Row Access Policy
CREATE AUTHENTICATION

Creates and enables an authentication method associated with users or roles. Vertica enables the authentication method automatically.

Syntax

```
CREATE AUTHENTICATION auth-method-name METHOD 'auth-type' acess-method
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth-method-name</td>
<td>Name of the authentication method to create, where <code>auth-method-name</code> conforms to conventions described in Identifiers.</td>
</tr>
<tr>
<td>auth-type</td>
<td>Name of the authentication method to use, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• gss</td>
</tr>
<tr>
<td></td>
<td>• ident</td>
</tr>
<tr>
<td></td>
<td>• ldap</td>
</tr>
<tr>
<td></td>
<td>• hash</td>
</tr>
<tr>
<td></td>
<td>• reject</td>
</tr>
<tr>
<td></td>
<td>• trust</td>
</tr>
<tr>
<td></td>
<td>• tls</td>
</tr>
<tr>
<td>acess-method</td>
<td>The access method the client uses to connect, specified in one of the following ways:</td>
</tr>
<tr>
<td></td>
<td>• LOCAL</td>
</tr>
<tr>
<td></td>
<td>Matches connection attempts made using local domain sockets.</td>
</tr>
<tr>
<td></td>
<td>• HOST [ TLS</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Matches connection attempts made using TCP/IP, where <code>host-ip-address</code> can be an IPv4 or IPv6 address. You can qualify HOST with one of the following options</td>
</tr>
<tr>
<td></td>
<td>- NO TLS: Match a plain (non-SSL/TLS) socket only.</td>
</tr>
</tbody>
</table>

**Privileges**

DBADMIN

**Examples**

See Creating Authentication Records.

**See Also**

- ALTER AUTHENTICATION
- DROP AUTHENTICATION
- GRANT (Authentication)
- REVOKE (Authentication)

**CREATE EXTERNAL TABLE AS COPY**

The CREATE EXTERNAL TABLE AS COPY statement creates a query definition for a table external to your Vertica database. This statement is a combination of the CREATE TABLE and COPY statements, supporting a subset of each statement's parameters, noted below. You can also use user-defined load extension functions (UDLs) to create external tables. For more information about UDL syntax, see User Defined Load (UDL).

You might discover that you need to change a column data type. For example, a VARCHAR might have longer data values than you anticipated at table-creation time. You can
use `ALTER TABLE` to change the data types of columns instead of dropping and recreating the table.

You can use `CREATE EXTERNAL TABLE AS COPY` with any types except types from the Place package.

**Note:** Vertica does not create superprojections for external tables, since external tables are not stored in the database.

### Syntax

```
CREATE EXTERNAL TABLE [ IF NOT EXISTS ] [[`database`.]`schema.`]`table-name`
... ( [ Column-Definition [ , ... ] )
[[INCLUDE | EXCLUDE] [SCHEMA] PRIVILEGES]
AS COPY
... [ { `column-as-expression` | `column` }
... [ { `DELIMITER` [ AS ] `char` ]
... [ { `ENCLOSED` [ BY ] `char` ]
... [ `ENFORCELENGTH` ]
... [ { `ESCAPE` [ AS ] `char` | NO ESCAPE ]
... [ { `FILLER` `datatype` ]
... [ { `FORMAT` `format` ]
... [ { `NULL` [ AS ] `string` ]
... [ { `TRIM` `byte` ]
... [ , ] ) }
... [ COLUMN OPTION ( `column` ]
... [ { `DELIMITER` [ AS ] `char` ]
... [ { `ENCLOSED` [ BY ] `char` ]
... [ `ENFORCELENGTH` ]
... [ { `ESCAPE` [ AS ] `char` | NO ESCAPE ]
... [ { `FORMAT` `format` ]
... [ { `NULL` [ AS ] `string` ]
... [ { `TRIM` `byte` ]
... [ , ] ) ]
{ FROM
... `pathToData` [ , ... ] [ ON `nodename` | ON ANY NODE | ON (nodeset) ]
... [ WITH ] SOURCE `source`([`arg=value` [, ... ] )
... [ `BZIP` | `GZIP` | `LZO` | UNCOMPRESSED ] [ , ... ]
... [ `NATIVE` ]
... [ `FIXEDWIDTH COLSIZES` `{ ( `integer` ) ,...,}` ]
... [ `NATIVE VARCHAR` ]
... [ ORC ]
... [ PARQUET ]
... ]
... [ `ABORT ON ERROR` ]
... [ `DELIMITER` [ AS ] `char` ]
... [ `ENCLOSED` BY `char` [ AND `char` ] ]
... [ `ENFORCELENGTH` ]
... [ `ERROR TOLERANCE` ]
... [ `ESCAPE` AS `char` | NO ESCAPE ]
... [ `EXCEPTIONS` `path` [ ON `nodename` ] [ , ... ] ]
... [ WITH ] [... [ FILTER `filter`([`arg=value` [, ... ] ) ] [ ... ]]
... [ NULL [ AS ] `string` ]
... [ WITH ] [ PARSER `parser`([`arg=value` [, ... ] ) ]
... [ RECORD TERMINATOR `string` ]
```
Parameters

For all supported parameters, see the CREATE TABLE and COPY statements. For information on using this statement with UDLs see Load (UDLs).

Privileges

You must be a database superuser to create external tables, unless you have created a user-accessible storage location to which the COPY refers, see CREATE LOCATION. If external tables exist, you must also be a database superuser to access them through a select statement.

You must have full access (including SELECT) to an external table that a user has privileges to create. The database superuser must also grant READ access to the USER-accessible storage location, see GRANT (Storage Location).

Examples

Examples of external table definitions:

```sql
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat' DELIMITER ',';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM 'hdfs:///dat/ext1.dat';
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat.bz2' BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE ext2 (x integer, y integer) AS COPY (x as '5', y) FROM '/tmp/ext1.dat.bz2'
   BZIP DELIMITER ',';
=> CREATE EXTERNAL TABLE sales (itemID INT, date DATE, price FLOAT)
   AS COPY FROM 's3://datalake/sales/*.parquet' PARQUET;
```

To allow users without superuser access to use these tables, create a location for 'user' usage and grant access to it. This example shows granting access to a user named Bob to any external table whose data is located under /tmp (including in subdirectories to any depth):

```sql
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';
=> GRANT ALL ON LOCATION '/tmp' to Bob;
```

The following example shows CREATE EXTERNAL TABLE using a source:
CREATE SOURCE curl AS LANGUAGE 'C++' NAME 'CurlSourceFactory' LIBRARY curllib;
CREATE EXTERNAL TABLE curl_table1 as COPY SOURCE CurlSourceFactory;

See Also
Creating External Tables in the Administrator's Guide

CREATE FAULT GROUP

Creates a fault group, which can contain the following:

- One or more nodes
- One or more child fault groups
- One or more nodes and one or more child fault groups

The CREATE FAULT GROUP statement creates an empty fault group. You must run the ALTER FAULT GROUP statement to add nodes or other fault groups to an existing fault group.

Syntax

CREATE FAULT GROUP name

Parameters

| name | The name of the fault group to create, unique among all fault groups, where name conforms to conventions described in Identifiers. |

Privileges

Superuser
Example

The following command creates a fault group called parent0:

```sql
=> CREATE FAULT GROUP parent0;
CREATE FAULT GROUP
```

To add nodes or other fault groups to the parent0 fault group, run the `ALTER FAULT GROUP` statement.

See Also

- `V_CATALOG.FAULT_GROUPS`
- `V_CATALOG.CLUSTER_LAYOUT`
- `Fault Groups`
- `High Availability With Fault Groups`

CREATE FLEX TABLE

Creates a flex table in the logical schema. Declaring columns (or other supported parameters) is optional. If you do not declare any column definitions, the statement creates two columns automatically:

- `__raw__` : A LONG VARBINARY type column to store any unstructured data you load. This column has a NOT NULL constraint by default.
- `__identity__` : An IDENTITY column. Flex tables use this value for segmentation and sorting, when no other column definition exists.

Additionally, creating any flex table results in three associated objects:

- A flex table (`flex_table`) named in this statement
- A related keys table, called `flex_table_keys`
- A related view, called `flex_table_view`
Both the flex table and its associated _keys table are required to use flex tables successfully. The _keys table and _view are subservient objects of the flex table. Neither can exist without the flex table.

For more details about creating and using flex tables, see Creating Flex Tables and other sections in Using Flex Tables.

CREATE FLEX TABLE supports many of the parameters available when creating columnar tables, but not all. This section presents the optional use of column definitions, and the subset of supported parameters.

You can also create flex external tables, with some syntactical variations, as described in CREATE FLEX EXTERNAL TABLE AS COPY.

You cannot partition a flex table on any virtual column (key).

**Note:** Vertica does not support flexible global temporary tables.

### Syntax

```
CREATE {FLEX | FLEXIBLE} TABLE [ IF NOT EXISTS ] [[database.]schema.]table-name {
... ( [ column-definition [ , ... ] ] )
... | [ table-constraint ( column_name, ... )]
... | [ column-name-list (create table) ]
}
...
[ INCLUDE | EXCLUDE] [SCHEMA] PRIVILEGES ]
...
[ ORDER BY table-column [ , ... ] ]
...
[ ENCODED BY column-definition [ , ... ] ]
...
[ hash-segmentation-clause ]
..... | UNSEGMENTED [ NODE node | ALL NODES ]
...
[ KSAFE [k_num] ]
...
[ PARTITION BY partition-expression]
...
[ AS SELECT (column-name-list) FROM (table-name) ]
```

### Parameters

See the CREATE TABLE statement for all parameter descriptions.

### Unsupported CREATE Flex Table Option

You cannot use the following options when creating a flex table:

```
... AS [COPY] [ [ AT EPOCH LATEST ] ... | [ AT TIME 'timestamp' ] ]
.....[/*+ direct */] query
... | [ LIKE [schema.]existing-table [ INCLUDING PROJECTIONS | EXCLUDING PROJECTIONS ] ]
```
Default Flex Table and Keys Table Projections

Vertica automatically creates superprojections for both the flex table and keys tables when you create them.

If you create a flex table with one or more of the ORDER BY, ENCODED BY, SEGMENTED BY, or KS/SAFE clauses, the clause information is used to create projections. If no clauses are in use, Vertica uses the following defaults for unspecified aspects:

<table>
<thead>
<tr>
<th>Table</th>
<th>order_by</th>
<th>encoded_by</th>
<th>Segmentation</th>
<th>Ksafe</th>
</tr>
</thead>
<tbody>
<tr>
<td>flexible table</td>
<td><strong>identity</strong></td>
<td>none</td>
<td>by hash <strong>identity</strong></td>
<td>1</td>
</tr>
<tr>
<td>keys_table</td>
<td>frequency</td>
<td>none</td>
<td>replicated/unsegmented all nodes</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: When you build a view for a flex table (see `BUILD_FLEXTABLE_VIEW`), the view is ordered by frequency, desc, and key_name.

Privileges

CREATE privileges on the table schema.

Examples

The following example creates a flex table named darkdata without specifying any column information. Vertica creates a default superprojection and buddy projection as part of creating the table:

```sql
=> CREATE FLEXIBLE TABLE darkdata();
CREATE TABLE
```

The following example creates a table called darkdata1 with one column definition (date_col). The statement specifies the partition by clause to partition the data by year. Vertica creates a default superprojection and buddy projections as part of creating the table:

```sql
=> CREATE FLEX TABLE darkdata1 (date_col date NOT NULL) partition by extract('year' from date_col);
CREATE TABLE
```
CREATE FLEX EXTERNAL TABLE AS COPY

The CREATE FLEX EXTERNAL TABLE AS COPY statement creates a flexible external table. This statement is a combination of the CREATE FLEX TABLE and COPY statements, supporting a subset of each statement's parameters, as noted below. You can also use user-defined load extension functions (UDLs) to create external flex tables. For more information about UDL syntax, see User Defined Load (UDL) and COPY. For more details about creating and using flex tables, see Using Flex Tables.

Note: Vertica does not create a superprojection for an external table when you create it.

For details about creating and using flex tables, see Creating Flex Tables in Using Flex Tables.

Syntax

CREATE {FLEX | FLEXIBLE} EXTERNAL TABLE [ IF NOT EXISTS ] [[database.]schema.]table-name 
... ( [ Column-Definition [ , ... ] ] )
) [ INCLUDE | EXCLUDE [SCHEMA] PRIVILEGES]
... AS COPY ... [ ( { column-as-expression | column } ......[ FILLER datatype ] ]
FROM ...
'pathToData' [ ON nodename | ON ANY NODE | ON (nodeset) ]
...... [ BZIP | GZIP | LZO | UNCOMPRESSED ] [, ...]
...[ WITH ]
...[ SOURCE source(arg='value')]
...[ FILTER filter(arg='value') ]
...[ PARSER flexparser(arg='value') ]
...[ DELIMITER [ AS ] 'char' ]
...[ TRAILING NULLCOLS ]
...[ NULL [ AS ] 'string' ]
...[ ESCAPE [ AS ] 'char' | NO ESCAPE ]
...[ ENCLOSED [ BY ] 'char' ]
...[ RECORD TERMINATOR 'string' ]
...[ SKIP integer ]
...[ SKIP BYTES integer ]
...[ TRIM 'byte' ]
...[ REJECTMAX integer ]
...[ EXCEPTIONS 'path' [ ON nodename ] [, ... ]]
...[ REJECTED DATA 'path' [ ON nodename ] [, ... ]]
Parameters

The following parameters from the parent statements are not supported in the CREATE FLEXIBLE EXTERNAL TABLE AS COPY statement:

<table>
<thead>
<tr>
<th>CREATE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS AT EPOCH LAST</td>
</tr>
<tr>
<td>AT TIME 'timestamp'</td>
</tr>
<tr>
<td>ORDER BY table-column [,... ]</td>
</tr>
<tr>
<td>ENCODED BY hash-segmentation-clause</td>
</tr>
<tr>
<td>UNSEGMENTED (node</td>
</tr>
<tr>
<td>KSAFE [k_num]</td>
</tr>
<tr>
<td>PARTITION BY partition-clause</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM STDIN</td>
</tr>
<tr>
<td>FROM LOCAL</td>
</tr>
<tr>
<td>DIRECT</td>
</tr>
<tr>
<td>TRICKLE</td>
</tr>
<tr>
<td>NO COMMIT</td>
</tr>
</tbody>
</table>

For all supported parameters, see the CREATE TABLE and COPY statements.

Privileges

Must be a database superuser to create external tables, unless the superuser has created a user-accessible storage location to which the COPY refers, as described in CREATE LOCATION. If external tables exist, you must also be a database superuser to access them through a select statement.

Permission requirements for flex external tables differ from other flex tables. You must have full access (including SELECT) to an external table that a user has privileges to create. The
database superuser must also grant READ access to the USER-accessible storage location, see **GRANT (Storage Location)**.

**Notes**

Canceling a CREATE FLEX EXTERNAL TABLE AS COPY statement can cause unpredictable results. Vertica recommends that you allow the statement to finish, then use **DROP TABLE** once the table exists.

**Examples**

To create an external flex table:

```sql
=> CREATE flex external table mountains() AS COPY FROM 'home/release/KData/kmm_-mountains.json' PARSER fjsonparser();
CREATE TABLE
```

As with other flex tables, creating an external flex table produces two regular tables: the named table and its associated _keys table. The keys table is not an external table:

```sql
=> \dt mountains
List of tables
  Schema | Name    | Kind | Owner | Comment
---------|---------|------|-------|---------
  public | mountains| table | release|         
(1 row)
```

You can use the helper function, **COMPUTE_FLEXTABLE KEYS_AND_BUILD VIEW**, to compute keys and create a view for the external table:

```sql
=> SELECT compute_flextable_keys_and_build_view ('appLog');

<table>
<thead>
<tr>
<th>compute_flextable_keys_and_build_view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Please see public.appLog_keys for updated keys</td>
</tr>
<tr>
<td>The view public.appLog_view is ready for querying</td>
</tr>
</tbody>
</table>
(1 row)
```

1. Check the keys from the _keys table for the results of running the helper application:

```sql
=> SELECT * FROM appLog_keys;

<table>
<thead>
<tr>
<th>key_name</th>
<th>frequency</th>
<th>data_type_guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>contributors</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>coordinates</td>
<td>8</td>
<td>varchar(20)</td>
</tr>
</tbody>
</table>
```
2. Query from the external flex table view:

```sql
=> SELECT "user.lang" FROM appLog_view;
user.lang  
----------
it  
en  
es  
en  
es  
tr  
en  
(12 rows)
```

See Also

- COPY Parameters
- CREATE EXTERNAL TABLE AS COPY
- CREATE TABLE
- CREATE FLEX TABLE
- SELECT

CREATE FUNCTION Statements

CREATE FUNCTION statements can create two different kinds of functions:

- User defined SQL functions let you define and store commonly-used SQL expressions as a function. User defined SQL functions are useful for executing complex queries and
combining Vertica built-in functions. You simply call the function name you assigned in your query.

- User defined scalar functions (UDSFs) take in a single row of data and return a single value. These functions can be used anywhere a native Vertica function or statement can be used, except CREATE TABLE with its PARTITION BY or any segmentation clause.

While you use CREATE FUNCTION to create both SQL and scalar functions, you use a different syntax for each function type. For more information, see:

- CREATE FUNCTION (SQL Functions)
- CREATE FUNCTION (UDF)

About Creating User Defined Transform Functions (UDTFs)

You can use a similar SQL statement to create user-defined transform functions. User defined transform functions (UDTFs) operate on table segments and return zero or more rows of data. The data they return can be an entirely new table, unrelated to the schema of the input table, including having its own ordering and segmentation expressions. They can only be used in a query’s SELECT list. For details about creating a UDTF, see CREATE TRANSFORM FUNCTION.

CREATE AGGREGATE FUNCTION

Adds a user-defined aggregate function (UDAF) stored in a shared Linux library to the catalog. You must have already loaded this library using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes data values to the code in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] AGGREGATE FUNCTION [[database.]schema.]function
  ... AS [LANGUAGE 'language'] NAME 'factory' LIBRARY library;
```

Parameters

| OR REPLACE | If you do not supply this parameter, CREATE AGGREGATE |
FUNCTION fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>function</td>
<td>Identifies the function to create, where function conforms to conventions described in Identifiers. Tip: This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.</td>
</tr>
<tr>
<td>LANGUAGE 'Language'</td>
<td>The language used to develop this function, currently C++ only. Default value: C++</td>
</tr>
<tr>
<td>NAME 'factory'</td>
<td>The name of the factory class in the shared library that generates the object to handle the function's processing.</td>
</tr>
<tr>
<td>LIBRARY Library</td>
<td>The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.</td>
</tr>
</tbody>
</table>

Notes

- The parameters and return value for the function are automatically determined by the CREATE AGGREGATE FUNCTION statement, based on data supplied by the factory class.

- When a User Defined Aggregate function that is defined multiple times with arguments of different data types is called, Vertica selects the function whose input parameters match the parameters in the function call to perform the processing.

- You can return a list of all SQL functions and User Defined Functions (including aggregates) by querying the system table V_CATALOG.USER_FUNCTIONS or executing the vsql meta-command df. Users see only the functions on which they have EXECUTE privileges.
Privileges

- Only a superuser can create or drop a User Defined Aggregate library.

- To create a User Defined Aggregate function, the user must have CREATE and USAGE privileges on the schema and USAGE privileges on the library.

- To use a User Defined Aggregate, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined function. See GRANT (User Defined Extension) and REVOKE (User Defined Extension).

Examples

The following example demonstrates loading a library named AggregateFunctions then defining a function named ag_avg and ag_cat that are mapped to the ag_cat AverageFactory and ConcatenateFactory classes in the library:

```sql
=> CREATE LIBRARY AggregateFunctions AS '/opt/vertica/sdk/examples/build/AggregateFunctions.so';
CREATE LIBRARY
=> CREATE AGGREGATE FUNCTION ag_avg AS LANGUAGE 'C++' NAME 'AverageFactory'
   library AggregateFunctions;
CREATE AGGREGATE FUNCTION
=> CREATE AGGREGATE FUNCTION ag_cat AS LANGUAGE 'C++' NAME 'ConcatenateFactory'
   library AggregateFunctions;
CREATE AGGREGATE FUNCTION
=> \x
Expanded display is on.
select * from user_functions;
-[
RECORD 1 ]-----------------------------------------------
schema_name | public
function_name | ag_avg
procedure_type | User Defined Aggregate
function_return_type | Numeric
function_argument_type | Numeric
function_definition | Class 'AverageFactory' in Library 'public.AggregateFunctions'
v volatility | |
isStrict | f
isFenced | f
comment | |
-[
RECORD 2 ]-----------------------------------------------
schema_name | public
function_name | ag_cat
procedure_type | User Defined Aggregate
function_return_type | Varchar
function_argument_type | Varchar
function_definition | Class 'ConcatenateFactory' in Library 'public.AggregateFunctions'
v volatility | |
isStrict | f
isFenced | f
```
CREATE ANALYTIC FUNCTION

Associates a User Defined Analytic Function (UDAnF) stored in a shared Linux library with a SQL function name. You must have already loaded the library containing the UDAnF using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes the arguments to the analytic function in the library to process.

Syntax

```
CREATE [ OR REPLACE ] ANALYTIC FUNCTION function-name
... AS [ LANGUAGE 'language' ] NAME 'factory'
... LIBRARY library_name
... [ FENCED | NOT FENCED ];
```

Parameters

<table>
<thead>
<tr>
<th>function-name</th>
<th>The name to assign to the UDAnF. This is the name you use in your SQL statements to call the function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANGUAGE 'language'</td>
<td>The language used to develop this function, one of the following:</td>
</tr>
</tbody>
</table>
### NAME 'factory'

The name of the C++ factory class in the shared library that generates the object to handle the function's processing.

### LIBRARY library-name

The name of the shared library that contains the C++ object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.

### FENCED | NOT FENCED

Enables or disables Fenced Mode for this function.

**Default Value:** FENCED

### Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.

- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.

- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

### Usage Considerations

- The parameters and return value for the function are automatically determined by the CREATE ANALYTIC FUNCTION statement, based on data supplied by the factory class.

- You can assign multiple functions the same name if they accept different sets of arguments. See Overloading Your UDX in Extending Vertica for more information.

- You can return a list of all UDFs by querying the system table V_CATALOG.USER_FUNCTIONS. Users see only the functions on which they have EXECUTE privileges.
Examples

This example shows how to create an analytic function named `an_rank` based on the factory class named `RankFactory` in the `AnalyticFunctions` library..

```sql
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'C++'
    NAME 'RankFactory' LIBRARY AnalyticFunctions;
```

See Also

Analytic Functions (UDAnFs)

CREATE FILTER

Adds a user-defined load filter function. You must have already loaded this library using the `CREATE LIBRARY` statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```sql
CREATE [ OR REPLACE ] FILTER [[database.]schema.]function
... AS [LANGUAGE 'language'] NAME 'factory' LIBRARY library
... [ FENCED | NOT FENCED ];
```

Parameters

<table>
<thead>
<tr>
<th>OR REPLACE</th>
<th>If you do not supply this parameter, the CREATE FILTER statement fails if an existing function matches the name and parameters of the filter function you are trying to define. If you do supply this parameter, the new filter function definition overwrites the old.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than <code>public</code>, you must supply the schema name. For example: <code>myschema.thisDbObject</code></td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x)  Page 3375 of 6180
If you specify a database, it must be the current database.

**function**

The name of the filter function to create. If the filter function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.

**LANGUAGE 'Language'**

The language used to develop this function, one of the following:

- C++ (default)
- Java

**NAME 'factory'**

The name of the factory class in the shared library that generates the object to handle the filter function's processing. This is the same name used by the RegisterFactory class.

**LIBRARY Library**

The name of the shared library that contains the C++ object to perform the processing for this filter function. This library must have been previously loaded using the CREATE LIBRARY statement.

**FENCED | NOT FENCED**

Enables or disables Fenced Mode for this function. Fenced mode is enabled by default.

### Privileges

- Only a superuser can create or drop a function that uses a U Dx library.

- To use a User Defined Filter, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined filter function. See GRANT (User Defined Extension) and REVOKE (User Defined Extension).

**Important:** Installing an untrusted UDL function can compromise the security of the server. U Dx's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to U Dx's to untrusted users.
Notes

- The parameters and return value for the filter function are automatically determined by the CREATE FILTER statement, based on data supplied by the factory class.

- You can return a list of all SQL functions and User Defined Functions by querying the system table V_CATALOG.USER_FUNCTIONs or executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Example

The following example demonstrates loading a library named iConverterLib, then defining a function named Iconverter that is mapped to the iConverterFactory factory class in the library:

```
=> CREATE LIBRARY iConverterLib as '/opt/vertica/sdk/examples/build/IconverterLib.so';
CREATE LIBRARY
=> CREATE FILTER Iconverter AS LANGUAGE 'C++' NAME 'IconverterFactory' LIBRARY IconverterLib;
CREATE FILTER FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
-[ RECORD 1 ]-----------------------------
schema_name | public
function_name | Iconverter
procedure_type | User Defined Filter
function_return_type |
function_argument_type |
function_definition |
v volatility |
is_strict | f
is_fenced | f
comment | |
```

See Also

- CREATE LIBRARY
- DROP FILTER
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
USER_FUNCTIONS

Load (UDLs)

CREATE FUNCTION (SQL Functions)

Lets you store SQL expressions as functions in Vertica for use in queries. These functions are useful for executing complex queries or combining Vertica built-in functions. You simply call the function name you assigned.

Note: This topic describes how to use CREATE FUNCTION to create a SQL function. If you want to create a user-defined scalar function (UDSF), see CREATE FUNCTION (UDF).

In addition, if you want to see how to create a user-defined transform function (UDTF), see CREATE TRANSFORM FUNCTION.

Syntax

CREATE [ OR REPLACE ] FUNCTION
... [[database.]schema.]function( [ argname argtype [, ...] ] )
... RETURN rettype
... AS
... BEGIN
...... RETURN expression;
... END;

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>function</td>
<td>Specifies a name for the SQL function to create, where function conforms to conventions described in Identifiers. When using more than one schema, specify the schema that contains the function, as noted above.</td>
</tr>
<tr>
<td>argname</td>
<td>Specifies the name of the argument.</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>argtype</td>
<td>Specifies the data type for argument that is passed to the function. Argument types must match Vertica type names. See SQL Data Types.</td>
</tr>
<tr>
<td>rettype</td>
<td>Specifies the data type to be returned by the function.</td>
</tr>
<tr>
<td>RETURN expression;</td>
<td>Specifies the SQL function (function body), which must be in the form of ‘RETURN expression.’ expression can contain built-in functions, operators, and argument names specified in the CREATE FUNCTION statement. A semicolon at the end of the expression is required.</td>
</tr>
</tbody>
</table>

**Note:** Only one RETURN expression is allowed in the CREATE FUNCTION definition. FROM, WHERE, GROUP BY, ORDER BY, LIMIT, aggregation, analytics, and meta function are not allowed.

### Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.

- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.

- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

See [GRANT (User Defined Extension)] and [REVOKE (User Defined Extension)].

### Notes

- A SQL function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

- SQL Macros are flattened in all cases, including DDL.
You can **create views** on the queries that use SQL functions and then query the views. When you create a view, a SQL function replaces a call to the user-defined function with the function body in a view definition. Therefore, when the body of the user-defined function is replaced, the view should also be replaced.

If you want to change the body of a SQL function, use the CREATE OR REPLACE syntax. The command replaces the function with the new definition. If you change only the argument name or argument type, the system maintains both versions under the same function name. See Examples section below.

If multiple SQL functions with same name and argument type are in the search path, the first match is used when the function is called.

The strictness and volatility (stable, immutable, or volatile) of a SQL Macro are automatically inferred from the function's definition. Vertica then determines the correctness of usage, such as where an immutable function is expected but a volatile function is provided.

You can return a list of all SQL functions by querying the system table `V_CATALOG.USER_FUNCTIONS` and executing the vsql meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

**Example**

This following statement creates a SQL function called `myzeroifnull` that accepts an INTEGER argument and returns an INTEGER result.

```sql
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
   AS BEGIN
       RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
   END;
```

You can use the new SQL function (`myzeroifnull`) anywhere you use an ordinary SQL expression. For example, create a simple table:

```sql
=> CREATE TABLE tabwnulls(col1 INT);
=> INSERT INTO tabwnulls VALUES(1);
=> INSERT INTO tabwnulls VALUES(NULL);
=> INSERT INTO tabwnulls VALUES(0);
=> SELECT * FROM tabwnulls;
```

```
---
1
0
(3 rows)
```
Use the myzeroifnull function in a SELECT statement, where the function calls col1 from table tabwnulls:

```sql
=> SELECT myzeroifnull(col1) FROM tabwnulls;
myzeroifnull
-------
 1
 0
 0
(3 rows)
```

Use the myzeroifnull function in the GROUP BY clause:

```sql
=> SELECT COUNT(*) FROM tabwnulls GROUP BY myzeroifnull(col1);
count
-------
 2
 1
(2 rows)
```

If you want to change a SQL function's body, use the CREATE OR REPLACE syntax. The following command modifies the CASE expression:

```sql
=> CREATE OR REPLACE FUNCTION zerowhennull(x INT) RETURN INT AS BEGIN
  RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
END;
```

To see how this information is stored in the Vertica catalog, see Viewing Information About SQL Functions in Extending Vertica.

See Also

- ALTER FUNCTION (UDF)
- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Using User-Defined SQL Functions
CREATE FUNCTION (UDF)

Adds a user-defined function (UDF) to the catalog. You must have already loaded this library using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Note: This topic describes how to use CREATE FUNCTION to create a User Defined Function. If you want to create a SQL function, see CREATE FUNCTION (SQL Function).

In addition, if you want to create a user-defined transform function (UDTF), see CREATE TRANSFORM FUNCTION.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [[database.]schema.]function
... AS [ LANGUAGE 'Language' ] NAME 'factory' LIBRARY Library
[ FENCED | NOT FENCED ];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>If you do not supply this parameter, the CREATE FUNCTION statement fails if an existing function matches the name and parameters of the function you are trying to define. If you do supply this parameter, the new function definition overwrites the old.</td>
</tr>
<tr>
<td>database</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td>schema</td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>function</td>
<td>The name of the function to create, where function conforms to conventions described in Identifiers. If the function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they</td>
</tr>
</tbody>
</table>
are the same or similar.

**LANGUAGE 'Language'**
The language used to develop this function, one of the following:
- C++
- Python
- Java
- R

**NAME 'factory'**
The name of the factory class in the shared library that generates the object to handle the function's processing.

**LIBRARY Library**
The name of the file that contains the C++ library, Python file, Java Jar file, or R functions file to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.

**FENCED | NOT FENCED**
Enables or disables Fenced Mode for this function. Fenced mode is enabled by default. Functions written in Java and R always run in fenced mode.

### Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.

- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.

- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

### Notes

- The parameters and return value for the function are automatically determined by the CREATE FUNCTION statement, based on data supplied by the factory class.
Multiple functions can share the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDF function whose input parameters match the parameters in the function call to perform the processing. This behavior is similar to having multiple signatures for a method or function in other programming languages.

You can return a list of all SQL functions and UDFs by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the `vsql` meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.

Examples

The following example demonstrates loading a library named scalarfunctions, then defining a function named Add2ints that is mapped to the Add2intsFactory factory class in the library:

```
=> CREATE LIBRARY ScalarFunctions AS '/opt/vertica/sdk/examples/build/ScalarFunctions.so';
CREATE LIBRARY

=> CREATE FUNCTION Add2Ints AS 'C++' NAME 'Add2IntsFactory' LIBRARY ScalarFunctions;
CREATE FUNCTION

=> \x
Expanded display is on.
=> SELECT * FROM USER_FUNCTIONS;

- [ RECORD 1 ]-+-----------------------------------------------+
  | schema_name | public |
  | function_name | Add2Ints |
  | procedure_type | User Defined Function |
  | function_return_type | Integer |
  | function_argument_type | Integer, Integer |
  | function_definition | Class 'Add2IntsFactory' in Library 'public.ScalarFunctions' |
  | volatility | volatile |
  | is_strict | f |
  | is_fenced | t |
  | comment | |

=> \x
Expanded display is off.
=> -- Try a simple call to the function
=> SELECT Add2Ints(23,19);
  Add2Ints
  --------
  42
(1 row)
```
See Also

- CREATE LIBRARY
- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Developing User-Defined Extensions (UDxs)

CREATE PARSER

Adds a user-defined load parser function. You must have already loaded this library using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] PARSER [[database.]schema.]function
... AS [ LANGUAGE 'language' ] NAME 'factory' LIBRARY library
... [ FENCED | NOT FENCED ];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>If you do not supply this parameter, the CREATE PARSER statement fails if an existing function matches the name and parameters of the parser function you are trying to define. If you do supply this parameter, the new parser function definition overwrites the old.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject</td>
</tr>
</tbody>
</table>
If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>function</th>
<th>The name of the parser function to create. If the parser function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.</th>
</tr>
</thead>
</table>
| LANGUAGE 'Language' | The language used to develop this function, one of the following:  
  * C++ (default)  
  * Java |
| NAME 'factory' | The name of the factory class in the shared library that generates the object to handle the parser function's processing. This is the same name used by the RegisterFactory class. |
| LIBRARY Library | The name of the shared library that contains the C++ object to perform the processing for this parser function. This library must have been previously loaded using the CREATE LIBRARY statement. |
| FENCED | NOT FENCED | Enables or disables Fenced Mode for this function.  
**Default Value:** FENCED |

### Privileges

- Only a superuser can create or drop a function that uses a UDx library.

- To use a User Defined Parser, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined parser function. See [GRANT (User Defined Extension)](User Defined Extension) and [REVOKE (User Defined Extension)](User Defined Extension).

---

**Important:** Installing an untrusted UDL function can compromise the security of the server. UDx's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDx's to untrusted users.
Notes

- The parameters and return value for the parser function are automatically determined by the CREATE PARSER statement, based on data supplied by the factory class.

- You can return a list of all SQL functions and User Defined Functions by querying the system table V_CATALOG.USER_FUNCTIONS or executing the vsq1 meta-command \df. Users see only the functions on which they have EXECUTE privileges.

Example

The following example demonstrates loading a library named BasicIntegrerParserLib, then defining a function named BasicIntegerParser that is mapped to the BasicIntegerParserFactory factory class in the library:

```
=> CREATE LIBRARY BasicIntegerParserLib as '/opt/vertica/sdk/examples/build/BasicIntegerParser.so';
CREATE LIBRARY
=> CREATE PARSER BasicIntegerParser AS LANGUAGE 'C++' NAME 'BasicIntegerParserFactory' LIBRARY BasicIntegerParserLib;
CREATE PARSER FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
-[ RECORD 1 ]------------------------
schema_name | public
function_name | BasicIntegerParser
procedure_type | User Defined Parser
function_return_type | 
function_argument_type |
function_definition |
v volatility |
is_strict | f
is_fenced | f
comment |
```

See Also

- CREATE LIBRARY
- DROP PARSER
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
USER_FUNCTIONS

Load (UDLs)

CREATE SOURCE

Adds a user-defined load source function. You must have already loaded this library using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes the parameters to the function in the library to process it.

Syntax

```
CREATE [ OR REPLACE ] SOURCE [[database.]schema.]function ...
    AS LANGUAGE 'language' NAME 'factory' LIBRARY library ...
    [ FENCED | NOT FENCED ];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>If you do not supply this parameter, the CREATE SOURCE statement fails if an existing function matches the name and parameters of the source function you are trying to define. If you do supply this parameter, the new source function definition overwrites the old.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td>function</td>
<td>The name of the source function to create. If the source function name is schema-qualified (as above), the function is created in the specified schema. This name does not need to match the name of the factory, but it is less confusing if they are the same or similar.</td>
</tr>
<tr>
<td>LANGUAGE 'language'</td>
<td>The language used to develop this function, one of the following:</td>
</tr>
</tbody>
</table>
### NAME 'factory'

The name of the factory class in the shared library that generates the object to handle the source function's processing.

This is the same name used by the RegisterFactory class.

### LIBRARY Library

The name of the shared library that contains the C++ object to perform the processing for this source function. This library must have been previously loaded using the `CREATE LIBRARY` statement.

### FENCED | NOT FENCED

Enables or disables Fenced Mode for this function.

**Default Value:** FENCED

---

**Privileges**

- Only a superuser can create or drop a function that uses a UDx library.

- To use a User Defined Source, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined source function. See [GRANT (User Defined Extension)](https://docs.vertica.com/vertica-analytic-database-manual/latest/functions/user-defined-source-create.html) and [REVOKE (User Defined Extension)](https://docs.vertica.com/vertica-analytic-database-manual/latest/functions/user-defined-source-create.html).

---

**Important:** Installing an untrusted UDL function can compromise the security of the server. UDx's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDx's to untrusted users.

---

**Notes**

- The parameters and return value for the source function are automatically determined by the `CREATE SOURCE` statement, based on data supplied by the factory class.

- You can return a list of all SQL functions and User Defined Functions by querying the system table `V_CATALOG.USER_FUNCTIONS` or executing the `vsql` meta-command `\df`. Users see only the functions on which they have EXECUTE privileges.
Example

The following example demonstrates loading a library named curlib, then defining a function named curl that is mapped to the CurlSourceFactory factory class in the library:

```sql
=> CREATE LIBRARY curlib as '/opt/vertica/sdk/examples/build/cURLLib.so';
CREATE LIBRARY
=> CREATE SOURCE curl AS LANGUAGE 'C++' NAME 'CurlSourceFactory' LIBRARY curlib;
CREATE SOURCE
=> \x
Expanded display is on.
=> SELECT * FROM user_functions;
- [ RECORD 1 ]- ---------------------------
   schema_name | public
   function_name | curl
   procedure_type | User Defined Source
   function_return_type | |
   function_argument_type | |
   function_definition | |
   volatility | |
   is_strict | f
   is_fenced | f
   comment | |
```

See Also

- CREATE LIBRARY
- DROP SOURCE
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Load (UDLs)

CREATE TRANSFORM FUNCTION

Adds a user-defined transform function (UDTF) stored in a shared Linux library to the catalog. You must have already loaded this library using the CREATE LIBRARY statement. When you call the SQL function, Vertica passes the input table to the transform function in the library to process.
Note: This topic describes how to create a UDTF. To create a user-defined function (UDF), see CREATE FUNCTION (UDF). To create a SQL function, see CREATE FUNCTION (SQL).

Syntax

CREATE [OR REPLACE] TRANSFORM FUNCTION function-name
AS [LANGUAGE 'language'] NAME 'factory'
LIBRARY library-name
[FENCED | NOT FENCED]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>If you omit this parameter, the CREATE TRANSFORM FUNCTION statement fails if an existing function matches the name and parameters of the function you are trying to define. If you supply this parameter, the new function definition overwrites the old one.</td>
</tr>
<tr>
<td>function-name</td>
<td>The name to assign to the UDTF. This is the name you use in your SQL statements to call the function.</td>
</tr>
<tr>
<td>LANGUAGE 'language'</td>
<td>The language used to develop this function, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• C++ (default)</td>
</tr>
<tr>
<td></td>
<td>• Java</td>
</tr>
<tr>
<td></td>
<td>• R</td>
</tr>
<tr>
<td></td>
<td>• PYTHON</td>
</tr>
<tr>
<td>NAME 'factory'</td>
<td>The name of the factory class or R factory function in the shared library that generates the object to handle the function's processing.</td>
</tr>
<tr>
<td>LIBRARY library-name</td>
<td>The name of the shared library that contains the object to perform the processing for this function. This library must have been previously loaded using the CREATE LIBRARY statement.</td>
</tr>
<tr>
<td>FENCED</td>
<td>NOT FENCED</td>
</tr>
</tbody>
</table>
written in R always run in fenced mode.

**Default Value:** FENCED

### Privileges

- To create a function, the user must have CREATE privilege on the schema to contain the function and USAGE privilege on the library containing the function.

- To use a function, the user must have USAGE privilege on the schema that contains the function and EXECUTE privileges on the function.

- To drop a function, the user must either be a superuser, the owner of the function, or the owner of the schema which contains the function.

### UDTF Query Restrictions

A query that includes a UDTF cannot contain:

- Any statements other than the `SELECT` statement containing the call to the UDTF and a `PARTITION BY` expression

- Any other analytic function

- A call to another UDTF

- A `TIMESERIES` clause

- A pattern matching clause

- A gap filling and interpolation clause

### Notes

- The parameters and return values for the function are automatically determined by the `CREATE TRANSFORM FUNCTION` statement, based on data supplied by the factory class.

- You can assign multiple functions the same name if they have different parameters. When you call a multiply-defined function, Vertica selects the UDTF function whose input parameters match the parameters in the function call to perform the processing. This
behavior is similar to having multiple signatures for a method or function in other programming languages.

- You can return a list of all UDTFs by querying the system table `V_CATALOG.USER_FUNCTIONS`. You can only see functions for which you have EXECUTE privileges.

Examples

This example shows how to add a UDTF to the catalog.

```sql
=> CREATE TRANSFORM FUNCTION transFunct AS LANGUAGE 'C++' NAME 'myFactory' LIBRARY myFunction;
```

See Also

- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Developing User-Defined Extensions (UDxs)

CREATE HCATALOG SCHEMA

Define a schema for data stored in a Hive data warehouse using the HCatalog Connector. For more information, see Using the HCatalog Connector in Integrating with Apache Hadoop.

Most of the optional parameters are read out of Hadoop configuration files if available. If you copied the Hadoop configuration files as described in Configuring Vertica for HCatalog, you can omit most parameters. By default this statement uses the values specified in those configuration files. If the configuration files are complete, the following is a valid statement:

```sql
=> CREATE HCATALOG SCHEMA hcat;
```

If a value is not specified in the configuration files and a default is shown in the parameter list, then that default value is used.
Some parameters apply only if you are using HiveServer2 (the default). Others apply only if you are using WebHCat, a legacy Hadoop service. When using HiveServer2, use HIVESERVER2_HOSTNAME to specify the server host. When using WebHCat, use WEBSERVICE_HOSTNAME to specify the server host.

If you need to use WebHCat you must also set the HCatalogConnectorUseHiveServer2 configuration parameter to 0. See Apache Hadoop Parameters.

After creating the schema, you can change many (but not all) parameters using ALTER HCATALOG SCHEMA.

Syntax

```sql
CREATE HCATALOG SCHEMA [IF NOT EXISTS] schemaName
  [AUTHORIZATION user-id]
  [WITH [param=value [, ...] ] ]
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[IF NOT EXISTS]</td>
<td>If given, the statement exits without an error when the schema named in <code>schemaName</code> already exists.</td>
</tr>
<tr>
<td><code>schemaName</code></td>
<td>The name of the schema to create in the Vertica catalog. The tables in the Hive database will be available through this schema.</td>
</tr>
<tr>
<td>AUTHORIZATION</td>
<td>The name of a Vertica account to own the schema being created. This parameter is ignored if Kerberos authentication is being used; in that case the current vsql user is used.</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| HOSTNAME   | The hostname, IP address, or URI of the database server that stores the Hive data warehouse's metastore information.  
If you specify this parameter and do not also specify PORT, then this value must be in the URI format used for hive.metastore.uris in hive- |
site.xml.
If the Hive metastore supports High Availability, you can specify a comma-separated list of URIs for this value.
If this value is not specified, hive-site.xml must be available.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT</td>
<td>The port number on which the metastore database is running. If you specify this parameter, you must also specify HOSTNAME and it must be a name or IP address (not a URI).</td>
</tr>
</tbody>
</table>
| HIVESERVER2_HOSTNAME          | The hostname or IP address of the HiveServer2 service. This parameter is optional if in hive-site.xml you set one of the following properties:  
                                 | - hive.server2.thrift.bind.host to a valid host  
                                 | - hive.server2.support.dynamic.service.discovery to true  
                                 | This parameter is ignored if you are using WebHCat. |
| WEBSERVICE_HOSTNAME           | The hostname or IP address of the WebHCat service, if using WebHCat instead of HiveServer2. If this value is not specified, webhcat-site.xml must be available. |
| WEBSERVICE_PORT               | The port number on which the WebHCat service is running, if using WebHCat instead of HiveServer2. If this value is not specified, webhcat-site.xml must be available. |
| WEBHDFS_ADDRESS               | The host and port ("host:port") for the WebHDFS service. This parameter is used only for reading ORC and Parquet files. If this value is not set, hdfs-site.xml must be available to read these file types through the HCatalog Connector. |
| HCATALOG_SCHEMA               | The name of the Hive schema or database that the Vertica schema is being mapped to. The default is `schemaName`. |
| HCATALOG_USER                 | The username of the HCatalog user to use when making calls to the HiveServer2 or WebHCat server. The default is the current database user. |
| HCATALOG_CONNECTION_TIMEOUT   | The number of seconds the HCatalog Connector waits for a successful connection to the HiveServer or WebHCat server. A value of 0 means wait indefinitely. |
| HCATALOG_                      | The lowest data transfer rate (in bytes per second) from the |
**SLOW_TRANSFER_LIMIT**
HiveServer2 or WebHCat server that the HCatalog Connector accepts. See HCATALOG_SLOW_TRANSFER_TIME for details.

**HCATALOG_SLOWTRANSFER_TIME**
The number of seconds the HCatalog Connector waits before enforcing the data transfer rate lower limit. After this time has passed, the HCatalog Connector tests whether the data transfer rate is at least as fast as the value set in HCATALOG_SLOW_TRANSFER_LIMIT. If it is not, then the HCatalog Connector breaks the connection and terminates the query.

**SSL_CONFIG**
The path of the Hadoop ssl-client.xml configuration file. This parameter is required if you are using HiveServer2 and it uses SSL wire encryption. This parameter is ignored if you are using WebHCat.

The default values for HCATALOG_CONNECTOR_TIMEOUT, HCATALOG_SLOW_TRANSFER_LIMIT, and HCATALOG_SLOW_TRANSFER_TIME are set by the database configuration parameters HCatConnectionTimeout, HCatSlowTransferLimit, and HCatSlowTransferTime. See Apache Hadoop Parameters in the Administrator's Guide for more information.

### Configuration Files
The HCatalog Connector uses the following values from the Hadoop configuration files if you do not override them when creating the schema.

<table>
<thead>
<tr>
<th>File</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive-site.xml</td>
<td>hive.server2.thrift.bind.host (used for HIVESERVER2_HOSTNAME)</td>
</tr>
<tr>
<td></td>
<td>hive.server2.thrift.port</td>
</tr>
<tr>
<td></td>
<td>hive.server2.transport.mode</td>
</tr>
<tr>
<td></td>
<td>hive.server2.authentication</td>
</tr>
<tr>
<td></td>
<td>hive.server2.authentication.kerberos.principal</td>
</tr>
<tr>
<td></td>
<td>hive.server2.support.dynamic.service.discovery</td>
</tr>
<tr>
<td></td>
<td>hive.zookeeper.quorum (used as HIVESERVER2_HOSTNAME if dynamic service discovery is enabled)</td>
</tr>
<tr>
<td>File</td>
<td>Properties</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>hive.zookeeper.client.port</td>
<td></td>
</tr>
<tr>
<td>hive.server2.zookeeper.namespace</td>
<td></td>
</tr>
<tr>
<td>hive.metastore.uris (used for HOSTNAME and PORT)</td>
<td></td>
</tr>
<tr>
<td>ssl-client.xml</td>
<td>ssl.client.truststore.location</td>
</tr>
<tr>
<td></td>
<td>ssl.client.truststore.password</td>
</tr>
</tbody>
</table>

**Privileges**

The user must be a superuser or be granted all permissions on the database to use this statement.

The user also requires access to Hive data in one of the following ways:

- Have USAGE permissions on `hcatalog_schema`, if Hive does not use an authorization service (Sentry or Ranger) to manage access.

- Have permission through an authorization service, if Hive uses it to manage access. In this case you must either set `EnableHCatImpersonation` to 0, to access data as the Vertica principal, or grant users access to the HDFS data. For Sentry, you can use ACL synchronization to manage HDFS access.

- Be the dbadmin user, with or without an authorization service.

**Examples**

The following example shows how to use CREATE HCATALOG SCHEMA to define a new schema for tables stored in a Hive database and then query the system tables that contain information about those tables:

```
=> CREATE HCATALOG SCHEMA hcat WITH HOSTNAME='hcathost' PORT=9083
   HCATALOG_SCHEMA='default' HIVESERVER2_HOSTNAME='hs.example.com'
   SSL_CONFIG='/etc/hadoop/conf/ssl-client.xml' HCATALOG_USER='admin';
CREATE SCHEMA
=> \\x
Expanded display is on.
```
The following example shows how to specify more than one metastore host.

```sql
=> CREATE HCATALOG SCHEMA hcat
    WITH HOSTNAME='thrift://node1.example.com:9083,thrift://node2.example.com:9083';
```
CREATE LIBRARY

Loads a library containing user defined extensions (UDxs) into the Vertica catalog. Vertica automatically distributes copies of the library file and any supporting libraries to all cluster nodes. UDxs defined in the catalog that reference the updated library automatically start using the new library file. Nodes that are down or added to the cluster also receive a copy of the updated library file as soon as they join the cluster.

Because libraries are added to the database catalog, they persist across database restarts. After loading a library in the catalog, you can use statements such as CREATE FUNCTION to define the extensions contained in the library. See Developing User-Defined Extensions (UDxs) in Extending Vertica for details.

Syntax

CREATE [OR REPLACE] LIBRARY
   [[database.]schema.]library
AS 'library-path'
   [ DEPENDS 'support-path' ]
   [ LANGUAGE 'language' ]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR REPLACE</td>
<td>Replaces the old library with the new one. If you do not supply this parameter, the CREATE LIBRARY statement fails when an existing library matches the name the library you are trying to define.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td>library</td>
<td>A name to assign to this library, where library conforms to conventions described in Identifiers. Use this name in a CREATE FUNCTION statement to enable user defined functions stored in</td>
</tr>
</tbody>
</table>
the library.

Tip: While not required, it is good practice to match *Library* to the library file name.

<table>
<thead>
<tr>
<th><strong>Library-path</strong></th>
<th>The absolute path and file name of the library to load. This file must be located in the initiator node file system.</th>
</tr>
</thead>
</table>
| **DEPENDS 'support-path'** | Indicates that the UDx library depends on one or more support libraries, where *support-path* specifies one or more absolute paths to the support libraries files, located in the initiator node's file system. You can specify multiple support paths as follows:  
   - Separate multiple paths with colons (:).  
   - Specify a directory that contains multiple libraries with an asterisk wildcard (*). For example: /home/mydir/mylibs/*  
   If your Java library depends on native libraries (SO files), use DEPENDS to specify the path and call `System.loadLibrary()` in your UDx to load the native libraries from that path. |
| **LANGUAGE 'Language'** | The programming language used to develop the function, where *Language* is one of the following:  
   - C++  
   - Python  
   - Java  
   - R |

**Privileges**

Superuser
Requirements

- Vertica makes its own copies of the library files. Later modification or deletion of the original files specified in the statement does not affect the library defined in the catalog. To update the library, use `ALTER LIBRARY`.

- Loading a library is no guarantee that it functions correctly. `CREATE LIBRARY` performs some basic checks on the library file to verify it is compatible with Vertica. The statement fails if it detects that the library was not correctly compiled or it finds other basic incompatibilities. However, `CREATE LIBRARY` cannot detect many other issues in shared libraries.

- You can choose to run UDxs developed in C++ in unfenced mode. Unfenced UDxs run directly in the Vertica process. If the U Dx you run in unfenced mode has bugs, it can negatively impact the database, causing instability or even crashes. To avoid these issues, run your U Dx only in fenced mode. A few U Dx types can only be run in unfenced mode.

Examples

To load a library in the home directory of the dbadmin account with the name MyFunctions:

```sql
=> CREATE LIBRARY MyFunctions AS 'home/dbadmin/my_functions.so';
```

To load a library located in the directory where you started `vsql`:

```sql
=> \set libfile '\''pwd'\''/MyOtherFunctions.so'';
=> CREATE LIBRARY MyOtherFunctions AS :libfile;
```

To load a Java library named `JavaLib.jar` that depends on multiple support JAR files in the `/home/dbadmin/mylibs` subdirectory:

```sql
=> CREATE LIBRARY DeleteVowelsLib AS `/home/dbadmin/JavaLib.jar`
   DEPENDS `'/home/dbadmin/mylibs/*' LANGUAGE 'JAVA';
```

See Also

- `DROP LIBRARY`
CREATE LOCAL TEMPORARY VIEW

Creates or replaces a local temporary view. Views are read only, so they do not support insert, update, delete, or copy operations. Local temporary views are session-scoped, so they are visible only to their creator in the current session. Vertica drops the view when the session ends.

**Note:** Vertica does not support global temporary views.

**Syntax**

```
CREATE [OR REPLACE] LOCAL TEMP[ORARY] VIEW view [ (column[, ...]) ] AS query
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OR REPLACE</strong></td>
<td>Specifies to overwrite the existing view <code>view-name</code>. If you omit this option and <code>view-name</code> already exists, <code>CREATE VIEW</code> returns an error.</td>
</tr>
<tr>
<td><strong>view</strong></td>
<td>Identifies the view to create, where <code>view-name</code> conforms to conventions described in <code>Identifiers</code>. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.</td>
</tr>
<tr>
<td><strong>column [ , ...]</strong></td>
<td>A list of names to use as view column names. Vertica maps view column names to query columns according to the order of their respective lists. By default, the view uses column names as they are specified in the query. Each view can contain up to 1600 columns.</td>
</tr>
<tr>
<td><strong>AS query</strong></td>
<td>A <code>SELECT</code> statement that the temporary view executes. The <code>SELECT</code> statement can reference tables, temporary tables, and other views.</td>
</tr>
</tbody>
</table>

**Privileges**

See **Creating Views**
Example

The following CREATE LOCAL TEMPORARY VIEW statement creates the temporary view myview. This view sums all individual incomes of customers listed in the store.store_sales_fact table, and groups results by state:

```sql
=> CREATE LOCAL TEMP VIEW myview AS
    SELECT SUM(annual_income), customer_state FROM public.customer_dimension
    WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact)
    GROUP BY customer_state
    ORDER BY customer_state ASC;
```

The following example uses the temporary view myview with a WHERE clause that limits the results to combined salaries greater than $2 billion:

```sql
=> SELECT * FROM myview WHERE SUM > 200000000;
```

<table>
<thead>
<tr>
<th>SUM</th>
<th>customer_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>2723441590</td>
<td>AZ</td>
</tr>
<tr>
<td>29253817091</td>
<td>CA</td>
</tr>
<tr>
<td>4907216137</td>
<td>CO</td>
</tr>
<tr>
<td>3769455689</td>
<td>CT</td>
</tr>
<tr>
<td>3330524215</td>
<td>FL</td>
</tr>
<tr>
<td>4581840709</td>
<td>IL</td>
</tr>
<tr>
<td>3310667307</td>
<td>IN</td>
</tr>
<tr>
<td>2793284639</td>
<td>MA</td>
</tr>
<tr>
<td>5225333668</td>
<td>MI</td>
</tr>
<tr>
<td>2128169759</td>
<td>NV</td>
</tr>
<tr>
<td>2886158503</td>
<td>PA</td>
</tr>
<tr>
<td>2832710696</td>
<td>TN</td>
</tr>
<tr>
<td>14215397659</td>
<td>TX</td>
</tr>
<tr>
<td>2642551509</td>
<td>UT</td>
</tr>
</tbody>
</table>

(14 rows)

See Also

- ALTER VIEW
- CREATE VIEW
- Creating Views
CREATE LOCATION

Creates a new storage location where Vertica can store data. After you create the location, you create storage policies that assign the storage location to the database objects that will store data in the location.

Cautions

While no technical issue prevents you from using CREATE LOCATION to add one or more Network File System (NFS) storage locations, Vertica does not support NFS data or catalog storage except for MapR mount points. You will be unable to run queries against any other NFS data. When creating locations on MapR file systems, you must specify ALL NODES SHARED.

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Syntax

CREATE LOCATION 'path'
   [NODE 'nodename' | ALL NODES]
   [SHARED]
   [USAGE 'usertype']
   [LABEL 'labelname']

Arguments

<table>
<thead>
<tr>
<th><strong>path</strong></th>
<th>Where Vertica will store this location's data. The type of filesystem on which the location is based determines the format of this argument:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- For storage locations on the Linux filesystem, <em>path</em> must be an absolute path to the directory where Vertica can write the storage location's data.</td>
</tr>
<tr>
<td></td>
<td>- For storage locations on HDFS, <em>path</em> must be an HDFS URL where</td>
</tr>
</tbody>
</table>
Vertica can write the storage location's data. See more information below.

- For USER storage locations on S3 in Eon Mode Beta only, *path* must be a URL of the form 's3://bucket/path'.

| **NODE 'nodename'** <br>**ALL NODES** | The node or nodes on which the storage location is defined.  
- **NODE**: Use this keyword to create the storage location on a single node. Specify the node using its name as it appears in the NODES system table.  
- **ALL NODES**: Use this keyword to create the storage location on all nodes  
  **Default Value**: ALL NODES |
| **SHARED** | Indicates the location set by the *path* is shared (used by all of the nodes) rather than local to each node. See below for details. |
| **USAGE 'use-type'** | The type of data the storage location can hold, where *use-type* is one of the following:  
- **TEMP**: Vertica uses the location to store temporary files it creates while processing queries.  
- **DATA**: The storage location can only store data.  
- **TEMP,DATA (default)**: The storage location can store both temporary files and data.  
- **USER**: Users who have been granted access to the storage location can read and store data there (see GRANT (Storage Location)). This usage type can be used with external tables (see CREATE EXTERNAL TABLE AS COPY), which by default are readable only by administrators.  
  **Default Value**: 'TEMP,DATA' |
| **LABEL 'Labelname'** | A label for the storage location. You use this name later when assigning the storage location to data objects. |
Privileges

Superuser

The Vertica process must also have read and write permissions to the location where date will be stored. Each type of filesystem has its own requirements:

- **Linux**: The database administrator account (usually named dbadmin) must have full read and write access to the directory in the *path* argument.

- **HDFS without Kerberos**: Requires a Hadoop user whose username matches the Vertica database administrator username (usually dbadmin). This Hadoop user must have read and write access to the HDFS directory specified in the *path* argument.

- **HDFS with Kerberos**: Requires a Hadoop user whose username matches the principal in the keytab file on each Vertica node. This is not the same as the database administrator username. This Hadoop user must have read and write access to the HDFS directory stored in the *path* argument.

Shared Versus Local Storage

The *shared* keyword indicates that the location set by the *path* argument is shared by all nodes. Most remote filesystems (such as HDFS) are shared. For these filesystems, the *path* argument represents a single location where all of the nodes store data. Each node creates its own subdirectory to hold its own files in a shared storage location. These subdirectories prevent the nodes from overwriting each other's files. Even if your cluster has only one node, you must include the *shared* keyword if you are using a remote filesystem. If the location is declared as USER Vertica does not create sub directories for each node. The setting of USER takes precedence over *shared*.

If you do not supply this keyword, the new storage location is local. The *path* argument specifies a location that is unique for each node in the cluster. This location is usually a path in the node's own filesystem. Storage locations contained in filesystems that are local to each node (such as the Linux filesystem) are always local.

HDFS URLs

To specify a path for a location on HDFS, use URLs in the *hdfs* scheme. In most cases you can use *hdfs://* (three slashes) followed by the HDFS path. To use HDFS URLs you must give
Vertica access to some HDFS configuration files. For more information, see Reading Directly from HDFS and Configuring the hdfs Scheme.

- Vertica node. This is not the same as the database administrator username. This Hadoop user must have read and write access to the HDFS directory stored in the path argument.

Examples

The following example shows how to create a storage location in the local Linux filesystem for temporary data storage.

```sql
=> CREATE LOCATION '/home/dbadmin/testloc' USAGE 'TEMP' LABEL 'tempfiles';
```

The following example shows how to create a storage location on HDFS in the /user/dbadmin directory. The HDFS cluster does not use Kerberos.

```sql
=> CREATE LOCATION 'hdfs:///user/dbadmin' ALL NODES SHARED
   USAGE 'data' LABEL 'coldstorage';
```

The following example shows how to create the same storage location, but on a Hadoop cluster that uses Kerberos. Note the output that reports the principal being used.

```sql
=> CREATE LOCATION 'hdfs:///user/dbadmin' ALL NODES SHARED
   USAGE 'data' LABEL 'coldstorage';
NOTICE 0: Performing HDFS operations using kerberos principal [vertica/hadoop.example.com]
CREATE LOCATION
```

The following example shows how to create a location for user data, grant access to it, and use it to create an external table.

```sql
=> CREATE LOCATION '/tmp' ALL NODES USAGE 'user';
CREATE LOCATION
=> GRANT ALL ON LOCATION '/tmp' to Bob;
GRANT PRIVILEGE
=> CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/data/ext1.dat' DELIMITER ',';
CREATE TABLE
```

For an example of a USER storage location using S3, see Browsing S3 Data Using External Tables in Using Eon Mode Beta.

See Also

- Managing Storage Locations in the Administrator's Guide
- Vertica Storage Location for HDFS in Integrating with Apache Hadoop.
CREATE NETWORK INTERFACE

Identifies a network interface to which a node belongs. Use this statement when you want to configure import/export operations from individual nodes to other Vertica clusters.

Syntax

```sql
CREATE NETWORK INTERFACE network-interface-name ON node-name WITH 'node-IP-address'
```

<table>
<thead>
<tr>
<th>network-interface-name</th>
<th>The name you assign to the network interface, where network-interface-name conforms to conventions described in Identifiers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-name</td>
<td>The name of the node.</td>
</tr>
<tr>
<td>node-IP-address</td>
<td>The node's IP address, either a public or private IP address. For more information, see Using Public and Private IP Networks.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

Examples

Create a network interface:

```sql
=> CREATE NETWORK INTERFACE mynetwork ON v_vmart_node0001 WITH '123.4.5.6';
```
CREATE NOTIFIER

Creates a push-based notifier to send event notifications and messages out of Vertica.

Syntax

```
CREATE NOTIFIER notifier-name ACTION action-url MAXMEMORYSIZE max-memory-size
... [ [NO] CHECK COMMITTED ]
... [ ENABLE | DISABLE ]
... [ IDENTIFIED BY uuid ]
... [ MAXPAYLOAD max-payload-size ]
... [ PARAMETERS 'adapter-params' ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>notifier-name</code></td>
<td>This notifier's unique identifier.</td>
</tr>
<tr>
<td><strong>ACTION action-url</strong></td>
<td>Identifies the target Kafka server, where <code>action-url</code> has the following format:</td>
</tr>
<tr>
<td></td>
<td>kafka://kafka-server-ip-address:port-number</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>kafka://127.0.0.1:9092</td>
</tr>
<tr>
<td><strong>MAXMEMORYSIZE</strong></td>
<td>The maximum size of the internal notifier, up to 2 TB, specified in kilobytes, megabytes,</td>
</tr>
<tr>
<td></td>
<td>gigabytes, or terabytes as follows:</td>
</tr>
<tr>
<td></td>
<td>MAXMEMORYSIZE integer(K</td>
</tr>
<tr>
<td></td>
<td>If the queue exceeds this size, the notifier drops excess messages.</td>
</tr>
<tr>
<td><strong>[NO] CHECK COMMITTED</strong></td>
<td>Specifies to wait for delivery confirmation before sending the next message in the queue.</td>
</tr>
<tr>
<td></td>
<td>Not all messaging systems support delivery confirmation.</td>
</tr>
<tr>
<td>**ENABLE</td>
<td>DISABLE**</td>
</tr>
<tr>
<td></td>
<td>this notifier to ENABLE.</td>
</tr>
<tr>
<td><strong>IDENTIFIED BY uuid</strong></td>
<td>Specifies the notifier's unique identifier. If set, all the</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| MAXPAYLOAD | The maximum size of the message, up to 2 TB, specified in kilobytes, megabytes, gigabytes, or terabytes as follows:
|            | MAXPAYLOAD integer{K|M|G|T}
|            | The default setting is adapter-specific—for example, 1 M for Kafka.
| PARAMETERS 'adapter-params' | Specifies one or more optional adapter parameters that are passed as a string to the adapter. Adapter parameters apply only to the adapter associated with the notifier.
|            | For Kafka notifiers, refer to Kafka and Vertica Configuration Settings.

**Privileges**

Database Administrator

**Examples**

Create a Kafka notifier:

```sql
=> CREATE NOTIFIER my_dc_notifier
    ACTION 'kafka://172.16.20.10:9092'
    MAXMEMORYSIZE '1G'
    IDENTIFIED BY 'f8b0278a-3282-4e1a-9c86-e0f3f042a971'
    NO CHECK COMMITTED;
```

Create a notifier with an adapter-specific parameter:

```sql
=> CREATE NOTIFIER my_notifier
    ACTION 'kafka://127.0.0.1:9092'
    MAXMEMORYSIZE '10M'
    PARAMETERS 'queue.buffering.max.ms=1000';
```

**See Also**

- ALTER NOTIFIER
- DROP NOTIFIER
• Monitoring Vertica Using Notifiers in the Administrator's Guide.

**CREATE PROCEDURE**

Adds an external procedure to Vertica. See Implementing External Procedures in Extending Vertica for more information about external procedures.

**Syntax**

```sql
CREATE PROCEDURE [[database.]schema.]procedure( 
  ... [ argname ] [ argtype [, ... ] ] ) 
  AS 'executable' 
  ... LANGUAGE 'language' 
  ... USER 'OS-user'
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `[database.]schema` | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:  
  ```sql
  myschema.thisDbObject
  ```  
  If you specify a database, it must be the current database. |
| `procedure`     | Specifies a name for the external procedure, where `procedure-name` conforms to conventions described in Identifiers. |
| `argname`       | Presents a descriptive argument name to provide a cue to procedure callers. |
| `argtype`       | Specifies the data type for argument(s) that will be passed to the procedure. Argument types must be one of the following Vertica type names: BIGINT, BOOLEAN, DECIMAL, DOUBLE PRECISION, FLOAT, FLOAT8, INT, INT8, INTEGER, MONEY, NUMBER, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR. |
| `executable`   | Specifies the executable program in the procedures directory. |
### Language

<table>
<thead>
<tr>
<th>Language</th>
<th>Specifies the procedure language. This parameter must be set to EXTERNAL.</th>
</tr>
</thead>
</table>

| USER | Specifies the user executed as. The user is the owner of the file. The external program must allow execute privileges for this user. The user cannot be root. |

### Privileges

CREATE privilege on the schema to contain the procedure.

### System Security

- A procedure file must be owned by the database administrator (OS account) or by a user in the same group as the administrator. (The procedure file owner cannot be root.) The procedure file must also have the set UID attribute enabled, and allow read and execute permission for the group.

- External procedures that you create with `CREATE PROCEDURE` are always run with Linux dbadmin privileges. If a dbadmin or pseudosuperuser grants a non-dbadmin permission to run a procedure using `GRANT (Procedure)`, be aware that the non-dbadmin user runs the procedure with full Linux dbadmin privileges.

### Examples

The following example shows how to create procedure `helloplanet` for external procedure file `helloplanet.sh`. This file accepts one varchar argument.

Create the file:

```bash
#!/bin/bash
echo "hello planet argument: $1" >> /tmp/myprocedure.log
```

Create the procedure with the following SQL:

```sql
=> CREATE PROCEDURE helloplanet(arg1 varchar) AS 'helloplanet.sh' LANGUAGE 'external' USER 'dbadmin';
```
CREATE PROFILE

Creates a profile that controls password requirements for users.

Syntax

CREATE PROFILE name LIMIT [ password-parameter setting ]...

password-parameter

PASSWORD_LIFE_TIME
PASSWORD_GRACE_TIME
FAILED_LOGIN_ATTEMPTS
PASSWORD_LOCK_TIME
PASSWORD_REUSE_MAX
PASSWORD_REUSE_TIME
PASSWORD_MAX_LENGTH
PASSWORD_MIN_LENGTH
PASSWORD_MINUTES
PASSWORD_MIN_UPPERCASE_LETTERS
PASSWORD_MIN_LOWERCASE_LETTERS
PASSWORD_MIN_DIGITS
PASSWORD_MIN_SYMBOLS

Parameters

Note: All parameters that are not explicitly set in a new profile are set to default, and inherit their settings from the default profile.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the profile to create, where name conforms to conventions described in Identifiers.</td>
</tr>
<tr>
<td>PASSWORD_LIFE_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PASSWORD_GRACE_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ≥ 1: The number of days a password can be used after it expires.</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: No grace period.</td>
</tr>
<tr>
<td>FAILED_LOGIN_ATTEMPTS</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ≥ 1: The number of consecutive failed login attempts Vertica allows before locking your account.</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: Vertica allows an unlimited number of failed login attempts.</td>
</tr>
<tr>
<td>PASSWORD_LOCK_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ≥ 1: The number of days your account is locked after too many failed login attempts. The account is automatically unlocked when the lock time elapses.</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: Account remains indefinitely inaccessible until a superuser manually unlocks it.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_MAX</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ≥ 1: The number of times you must change your password before you can reuse an earlier password.</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: You can reuse an earlier password without any intervening changes.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_TIME</td>
<td>Set to an integer value, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ≥ 1: The number of days that must pass after a password is set before you can reuse it.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PASSWORD_MAX_LENGTH</td>
<td>The maximum number of characters allowed in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 8 and 100, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: Maximum of 100 characters</td>
</tr>
<tr>
<td>PASSWORD_MIN_LENGTH</td>
<td>The minimum number of characters required in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• 0 to PASSWORD_MAX_LENGTH</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: Minimum of PASSWORD_MAX_LENGTH</td>
</tr>
<tr>
<td>PASSWORD_MIN_LETTERS</td>
<td>Minimum number of letters (a-z and A-Z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_UPPERCASE_LETTERS</td>
<td>Minimum number of uppercase letters (A-Z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_LOWERCASE_LETTERS</td>
<td>Minimum number of lowercase letters (a-z) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>PASSWORD_MIN_DIGITS</td>
<td>Minimum number of digits (0-9) that must be in a password, one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Integer between 0 and PASSWORD_MAX_LENGTH, inclusive</td>
</tr>
<tr>
<td></td>
<td>• UNLIMITED: 0 (no minimum)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| PASSWORD_MIN_SYMBOLS                      | Minimum number of symbols—printable non-letter and non-digit characters such as $, #, @—that must be in a password, one of the following:  
  • Integer between 0 and PASSWORD_MAX_LENGTH, inclusive  
  • UNLIMITED: 0 (no minimum)                                                                                                           |

**Privileges**

Superuser

**Profile Settings and Client Authentication**

The following profile settings affect client authentication methods, such as LDAP or GSS:

- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME

All other profile settings are used only by Vertica to manage its passwords.

**Example**

```sql
=> CREATE PROFILE sample_profile LIMIT PASSWORD_MAX_LENGTH 20;
```

**See Also**

- ALTER PROFILE
- DROP PROFILE
- Creating a Database Name and Password
CREATE PROJECTION

Creates metadata for a projection in the Vertica catalog.

**Note:** For detailed information about using CREATE PROJECTION to create live aggregate projections and Top-K projections, see CREATE PROJECTION (Live Aggregate Projections). To create live aggregate projections that support user-defined transform functions, see CREATE PROJECTION (UDTFs).

**Syntax**

```
CREATE PROJECTION [ IF NOT EXISTS ] [[database.]schema.]projection

...[ ( 

........{ projection-col | grouped-clause

........ [ ENCODING encoding-type ]

........ [ ACCESSRANK integer ]

........}[,...]

...)

AS SELECT

...select-list from-clause

...[ ORDER BY column-expr[,...] ]

...[ segmentation-spec ]

...[ KSAFE [ k-num ] ]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF NOT EXISTS</strong></td>
<td>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created. The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does. For related information, see <strong>ON_ERROR_STOP</strong>.</td>
</tr>
<tr>
<td><strong>[database.]schema</strong></td>
<td>Specifies a schema. The projection and its anchor table must be in the same schema. If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>
| **projection** | Identifies the projection to create, where *projection* conforms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. The projection is created in the same schema as the anchor table.

If the projection is segmented, Vertica uses this string as the projection base name when it creates unique identifiers for buddy projections. For more information, see Projection Naming in the Administrator's Guide. |
|---------------------------------------------------------------|
| **projection-col** | The name of a projection column. The list of projection columns must match the *select-list* columns and expressions in number, type, and sequence.

If projection column names are omitted, Vertica uses the anchor table column names specified in *select-list*. |
| **grouped-clause** | See GROUPED Clause. |
| **ENCODING** **encoding-type** | Specifies the column encoding type, by default set to AUTO. |
| **ACCESSRANK** **integer** | Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking. |
| **select-list** | Specifies the columns or column expressions to select from one or more tables, one of the following:

- * (asterisk)
  Lists all columns in the queried tables.

- *expression [ [AS] output-name] }[,...]*
  A table column or column expression to select from the queried tables. You can optionally qualify *expression* with an output name, which can be used in two ways:

  - Label the column for display.
  - Refer to the column in the projection's ORDER BY clause. |
| **from-clause** | A comma-separated list of data sources to query. |
| ORDER BY | Specifies columns from the SELECT list on which to sort the projection. The ORDER BY clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order. If you omit the ORDER BY clause, Vertica uses **select-list** to sort the projection. |
| segmentation-spec | Specifies how to distribute projection data with one of the following clauses:  
- **hash-segmentation-clause**: Specifies to segment projection data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables.  
- **unsegmented-clause**: Specifies to create an unsegmented projection.  
If the anchor table and projection both omit specifying segmentation, the projection is defined with a hash segmentation clause that includes all columns in the SELECT list, as follows:  
`SEGMENTED BY HASH(column-expr[,...]) ALL NODES OFFSET 0;` |
| KSAFE `[k-num]` | Specifies K-safety for the projection, where `k-num` must be equal to or greater than system K-safety. Vertica ignores this parameter if set for unsegmented projections. If you omit `k-num`, Vertica uses system K-safety.  
Vertica sets projection K-safety as follows:  
- KSAFE and OFFSET clause omitted: Uses system K-safety.  
- OFFSET clause is omitted and KSAFE is specified: Uses KSAFE setting.  
- KSAFE is omitted and the OFFSET clause is specified: Uses OFFSET setting.  
If the CREATE PROJECTION statement specifies KSAFE and the OFFSET clause, Vertica returns an error. For general information, see K-Safety in Vertica Concepts. |
Privileges

Superuser, or the following:

- Anchor table owner
- CREATE privilege on the schema

Requirements

- To prevent data loss and inconsistencies, tables must have at least one superprojection. You cannot drop a projection if that projection is the table's only superprojection.
- You cannot drop a buddy projection if dropping that projection violates system K-safety.

Creating Projections with Expressions

The following example shows a projection that calculates the product of two numbers. The anchor table is defined as follows:

```sql
=> CREATE TABLE values (a INT, b INT);
```

To create a projection that calculates the product of a and b, use a statement like the following:

```sql
=> CREATE PROJECTION values_product (a, b, product_value) 
  AS SELECT a, b, a*b FROM values 
  SEGMENTED BY HASH(a) ALL NODES KSAFE;
```

To query that projection, you must use the name that Vertica assigned to it:

```sql
=> SELECT * FROM values_product_b0;
```

or

```sql
=> SELECT * FROM values_product_b1;
```
Grouping Correlated Columns

The following example shows how to group highly correlated columns bid and ask. The stock column is stored separately.

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION tradeproj (stock ENCODING RLE,
    GROUPED(bid ENCODING DELTAVAL, ask))
   AS (SELECT * FROM trades) KSAFE 1;
```

The following example shows how to create a projection that uses expressions in the column definition. The projection contains two integer columns a and b, and a third column product_value that stores the product of a and b:

```
=> CREATE TABLE values (a INT, b INT)
=> CREATE PROJECTION product (a, b, product_value) AS
   SELECT a, b, a*b FROM values ORDER BY a KSAFE;
```

See Also

Working with Projections

Encoding Types

Vertica supports various encoding and compression types, specified by the following ENCODING parameter arguments:

- AUTO (default)
- BLOCK_DICT
- BLOCKDICT_COMP
- BZIP_COMP
- COMMONDELTA_COMP
- DELTARANGE_COMP
- DELTAVAL
AUTO (default)

AUTO encoding is ideal for sorted, many-valued columns such as primary keys. It is also suitable for general purpose applications for which no other encoding or compression scheme is applicable. Therefore, it serves as the default if no encoding/compression is specified.

<table>
<thead>
<tr>
<th>Column data type</th>
<th>Default encoding type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY/VARBINARY</td>
<td>Lempel-Ziv-Oberhumer-based (LZO) compression</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>CHAR/VARCHAR</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>DATE/TIME/TIMESTAMP</td>
<td>Compression scheme based on the delta between consecutive column values.</td>
</tr>
<tr>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
</tr>
</tbody>
</table>

The CPU requirements for this type are relatively small. In the worst case, data might expand by eight percent (8%) for LZO and twenty percent (20%) for integer data.

BLOCK_DICT

For each block of storage, Vertica compiles distinct column values into a dictionary and then stores the dictionary and a list of indexes to represent the data block.

BLOCK_DICT is ideal for few-valued, unsorted columns where saving space is more important than encoding speed. Certain kinds of data, such as stock prices, are typically few-valued within a localized area after the data is sorted, such as by stock symbol and timestamp, and are good candidates for BLOCK_DICT. By contrast, long CHAR/VARCHAR columns are not good candidates for BLOCK_DICT encoding.

CHAR and VARCHAR columns that contain 0x00 or 0xFF characters should not be encoded with BLOCK_DICT. Also, BINARY/VARBINARY columns do not support BLOCK_DICT encoding.

BLOCK_DICT encoding requires significantly higher CPU usage than default encoding schemes. The maximum data expansion is eight percent (8%).
**BLOCKDICT_COMP**

This encoding type is similar to BLOCK_DICT except dictionary indexes are entropy coded. This encoding type requires significantly more CPU time to encode and decode and has a poorer worst-case performance. However, if the distribution of values is extremely skewed, using BLOCK_DICT_COMP encoding can lead to space savings.

**BZIP_COMP**

BZIP_COMP encoding uses the bzip2 compression algorithm on the block contents. See [bzip](https://bzip.org/) web site for more information. This algorithm results in higher compression than the automatic LZO and gzip encoding; however, it requires more CPU time to compress. This algorithm is best used on large string columns such as VARCHAR, VARBINARY, CHAR, and BINARY. Choose this encoding type when you are willing to trade slower load speeds for higher data compression.

**COMMONDELTA_COMP**

This compression scheme builds a dictionary of all deltas in the block and then stores indexes into the delta dictionary using entropy coding.

This scheme is ideal for sorted FLOAT and INTEGER-based (DATE/TIME/TIMESTAMP/INTERVAL) data columns with predictable sequences and only occasional sequence breaks, such as timestamps recorded at periodic intervals or primary keys. For example, the following sequence compresses well: 300, 600, 900, 1200, 1500, 600, 1200, 1800, 2400. The following sequence does not compress well: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55.

If delta distribution is excellent, columns can be stored in less than one bit per row. However, this scheme is very CPU intensive. If you use this scheme on data with arbitrary deltas, it can cause significant data expansion.

**DELTARANGE_COMP**

This compression scheme is primarily used for floating-point data; it stores each value as a delta from the previous one.

This scheme is ideal for many-valued FLOAT columns that are sorted or confined to a range. Do not use this scheme for unsorted columns that contain NULL values, as the storage cost for representing a NULL value is high. This scheme has a high cost for both compression and decompression.
To determine if DELTARANGE_COMP is suitable for a particular set of data, compare it to other schemes. Be sure to use the same sort order as the projection, and select sample data that will be stored consecutively in the database.

**DELTAVAL**

For INTEGER and DATE/TIME/TIMESTAMP/INTERVAL columns, data is recorded as a difference from the smallest value in the data block. This encoding has no effect on other data types.

DELTAVAL is best used for many-valued, unsorted integer or integer-based columns. CPU requirements for this encoding type are minimal, and data never expands.

**GCDDELTA**

For INTEGER and DATE/TIME/TIMESTAMP/INTERVAL columns, and NUMERIC columns with 18 or fewer digits, data is recorded as the difference from the smallest value in the data block divided by the greatest common divisor (GCD) of all entries in the block. This encoding has no effect on other data types.

ENCODING GCDDELTA is best used for many-valued, unsorted, integer columns or integer-based columns, when the values are a multiple of a common factor. For example, timestamps are stored internally in microseconds, so data that is only precise to the millisecond are all multiples of 1000. The CPU requirements for decoding GCDDELTA encoding are minimal, and the data never expands, but GCDDELTA may take more encoding time than DELTAVAL.

**GZIP_COMP**

This encoding type uses the gzip compression algorithm. See gzip web site for more information. This algorithm results in better compression than the automatic LZO compression, but lower compression than BZIP_COMP. It requires more CPU time to compress than LZO but less CPU time than BZIP_COMP. This algorithm is best used on large string columns such as VARCHAR, VARBINARY, CHAR, and BINARY. Use this encoding when you want a better compression than LZO, but at less CPU time than bzip2.

**RLE**

RLE (run length encoding) replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, it is best used for low cardinality columns that are present in the ORDER BY clause of a projection.
The Vertica execution engine processes RLE encoding run-by-run and the Vertica optimizer gives it preference. Use it only when run length is large, such as when low-cardinality columns are sorted.

The storage for RLE and AUTO encoding of CHAR/VARCHAR and BINARY/VARBINARY is always the same.

Valid Encoding for Numeric Data Types

- Valid encoding types for numeric data type columns with precision <= 18 include AUTO, BLOCK_DICT, BLOCKDICT_COMP, COMMONDELTA_COMP, DELTAVAL, GCDDELTA, and RLE.
- Valid encoding types for numeric data type columns with precision > 18 include AUTO, BLOCK_DICT, BLOCKDICT_COMP, RLE.
- For information on numeric data types, see the section, Numeric Data Types.

GROUPED Clause

Groups two or more columns into a single disk file. This minimizes file I/O for work loads that:

- Read a large percentage of the columns in a table.
- Perform single row look-ups.
- Query against many small columns.
- Frequently update data in these columns.

If you have data that is always accessed together and it is not used in predicates, you can increase query performance by grouping these columns. Once grouped, queries can no longer independently retrieve from disk all records for an individual column independent of the other columns within the group.

**Note:** RLE encoding is reduced when an RLE column is grouped with one or more non-RLE columns.

When grouping columns you can:

- Group some of the columns:
  
  (a, GROUPED(b, c), d)
• Group all of the columns:

\[(\text{GROUPED}(a, b, c, d))\]

• Create multiple groupings in the same projection:

\[(\text{GROUPED}(a, b), \text{GROUPED}(c, d))\]

**Note:** Vertica performs dynamic column grouping. For example, to provide better read and write efficiency for small loads, Vertica ignores any projection-defined column grouping (or lack thereof) and groups all columns together by default.

## Hash Segmentation Clause

Specifies how to segment projection data for distribution across some or all cluster nodes. You can specify segmentation for a table and a projection. If a table definition specifies segmentation, Vertica uses it for that table's auto-projections.

It is strongly recommended that you use Vertica's built-in **HASH** function, which distributes data evenly across the cluster, and facilitates optimal query execution.

### Syntax

```sql
SEGMEMTED BY expression { ALL NODES [ OFFSET offset ] | NODES node [ ,... ] }
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SEGMENTED BY</strong></td>
<td>A general SQL expression. Hash segmentation is the preferred method of segmentation. Vertica recommends using its built-in <strong>HASH</strong> function, whose arguments resolve to table columns. If you use an expression other than <strong>HASH</strong>, Vertica issues a warning. The segmentation expression should specify columns with a large number of unique data values and acceptable skew in their data distribution. In general, primary key columns that meet these criteria are good candidates for hash segmentation. For details, see <strong>Expression Requirements</strong> below.</td>
</tr>
</tbody>
</table>

**Expression Requirements**

- The expression should specify columns with a large number of unique data values and acceptable skew in their data distribution.
- In general, primary key columns that meet these criteria are good candidates for hash segmentation.
**Expression Requirements**

A segmentation expression must specify table columns as they are defined in the source table. Projection column names are not supported.

The following restrictions apply to segmentation expressions:

- All leaf expressions must be constants or column references to a column in the CREATE PROJECTION's SELECT list.
- The expression must return the same value over the life of the database.
- Aggregate functions are not allowed.
- The expression must return non-negative INTEGER values in the range $0 \leq x < 2^{63}$, and values are generally distributed uniformly over that range.

**Note:** If the expression produces a value outside the expected range—for example, a negative value—no error occurs, and the row is added to the projection's first segment.

**Examples**

The following CREATE PROJECTION statement creates projection `public.employee_dimension_super`. It specifies to include all columns in table `public.employee_dimension`. The hash segmentation clause invokes the Vertica HASH function to segment projection data on the column `employee_key`; it also includes the ALL NODES clause, which specifies to distribute projection data evenly across all nodes in the cluster:
CREATE PROJECTION public.employee_dimension_super
  AS SELECT * FROM public.employee_dimension
  ORDER BY employee_key
  SEGMENTED BY hash(employee_key) ALL NODES;

Unsegmented Clause

Specifies to distribute identical copies of table or projection data across the cluster. Use this clause to facilitate distributed query execution on tables and projections that are too small to benefit from segmentation.

Vertica uses the same name to identify all instances of an unsegmented projection. For more information about projection name conventions, see Projection Naming.

Syntax

UNSEGMENTED {NODE node | ALL NODES}

Parameters

| NODE node | Creates an unsegmented projection only on the specified node node. To obtain a list of all cluster nodes, query the system table V_CATALOG.NODES. |
| ALL NODES | Creates instances of the unsegmented projection on all cluster nodes. You must set this option if projection or system K-safety is greater than 0, otherwise Vertica regards the projection as unsafe and does not use it. |

Example

This example creates an unsegmented projection for table store.store_dimension:

=> CREATE PROJECTION store.store_dimension_proj (storekey, name, city, state)
  AS SELECT store_key, store_name, store_city, store_state
  FROM store.store_dimension
  UNSEGMENTED ALL NODES;

CREATE PROJECTION

=> SELECT anchor_table_name anchor_table, projection_name, node_name
  FROM PROJECTIONS WHERE projection_basename='store_dimension_proj';
anchor_table | projection_name | node_name
CREATE PROJECTION (Live Aggregate Projections)

Creates metadata for live aggregate projections in the Vertica catalog. Top-K projections are a type of live aggregate projection.

Information here focuses on creating live aggregate projections. For details about creating other types of projections, including projections with expressions, see CREATE PROJECTION.

Syntax

Grouping aggregate function results

CREATE PROJECTION [ IF NOT EXISTS ] [[database.]schema.]projection
... [ ( 
........{ projection-col | grouped-clause
........ [ ENCODING encoding-type ]
........ [ ACCESSRANK integer ]
........ )[,... ]
... ) ]
... GROUP BY column-expr
... [ KSAFE [ k-num ] ]

Top-K aggregation

CREATE PROJECTION [ IF NOT EXISTS ] projection-name
... [ ( 
........{ projection-col | grouped-clause
........ [ ENCODING encoding-type ]
........ [ ACCESSRANK integer ]
........ ) [,,... ]
...... )
... ]
... LIMIT num-rows OVER (PARTITION BY column-expr ORDER BY column-expr)
... [ KSAFE [ k-num ] ]
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF NOT EXISTS</td>
<td>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created. The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does. For related information, see ON_ERROR_STOP.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema. The projection and its anchor table must be in the same schema. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>projection</td>
<td>Identifies the projection to create, where projection conforms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. The projection is created in the same schema as the anchor table.</td>
</tr>
<tr>
<td>projection-col</td>
<td>The name of a projection column. If you do not specify projection column names, Vertica uses the anchor table column names in the SELECT statement.</td>
</tr>
<tr>
<td>grouped-clause</td>
<td>See GROUPED Clause.</td>
</tr>
<tr>
<td>ENCODING encoding-type</td>
<td>Specifies the column encoding type, by default set to AUTO.</td>
</tr>
<tr>
<td>ACCESSRANK integer</td>
<td>Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking.</td>
</tr>
<tr>
<td>table-col expr-with-table-cols</td>
<td>A table column or expression of table columns to be included in the projection. If you specify projection column names, the two lists of projection columns and table columns/expressions must exactly match in number and...</td>
</tr>
</tbody>
</table>
A schema table with the columns to include in the projection, as follows:

```
table-name [ AS ] alias [ ( column-alias [, ...] ) ]
```

One or more column expressions from the SELECT list. The first `column-expr` must be the first column expression in the SELECT list, the second `column-expr` must be the second column expression in the SELECT list, and so on.

The number of rows to return from the specified partition.

Specifies window partitioning by one or more column expressions from the SELECT list. The first `column-expr` is the first column expression in the SELECT list, the second `column-expr` is the second column expression in the SELECT list, and so on.

The order in which the top `k` rows are returned, by default in ascending (ASC) order. All column expressions must be from the SELECT list, where the first `column-expr` must be the first column expression in the SELECT list to follow the last `PARTITION BY` column expression.

Top-K projections support `ORDER BY NULLS FIRST/LAST`.

Specifies K-safety for the projection, where `k-num` must be equal to or greater than system K-safety. Vertica ignores this parameter if set for unsegmented projections. If you omit `k-num`, Vertica uses system K-safety.

Vertica sets projection K-safety as follows:

- **KSAFE and OFFSET clause omitted**: Uses system K-safety.
- **OFFSET clause is omitted and KSAFE is specified**: Uses KSAFE setting.
- **KSAFE is omitted and the OFFSET clause is specified**: Uses OFFSET setting.

If the `CREATE PROJECTION` statement specifies KSAFE and the OFFSET clause, Vertica returns an error.
For general information, see K-Safety in Vertica Concepts.

Privileges

Superuser, or the following:

- Anchor table owner
- CREATE privilege on the schema

Requirements and Restrictions

See:

- Creating Live Aggregate Projections
- Creating Top-K Projections

Examples

See:

- Live Aggregate Projection Example
- Top-K Projection Examples

CREATE PROJECTION (UDTFs)

Creates metadata in the Vertica catalog for projections that invoke user-defined transform functions (UDTFs).

Important: Currently, live aggregate projections can only reference UDTFs that are developed in C++.
Syntax

CREATE PROJECTION [ IF NOT EXISTS ] [[database.]schema.]projection
...[
........{ projection-col | grouped-clause
........ [ ENCODING encoding-type ]
........ [ ACCESSRANK integer ]
........} [, ... ]
...] ]
AS {
........batch-query FROM { prepass-query sq-results | table-ref }
........| prepass-query
...}

batch-query

SELECT { table-column | expr-with-table-cols }[,...], batch-udtf(batch-args)
...OVER (PARTITION BATCH BY partition-col-expr[,...] )
...[ AS (batch-output-cols) ]

prepass-query

SELECT { table-col | expr-with-table-cols }[,...], prepass-udtf(prepass-args)
...OVER (PARTITION PREPASS BY partition-col-expr[,...] )
...[ AS (prepass-output-cols) ] FROM table-ref

Parameters

<table>
<thead>
<tr>
<th>IF NOT EXISTS</th>
<th>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created. The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does. For related information, see ON_ERROR_STOP.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema. The projection and its anchor table must be in the same schema. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>projection</td>
<td>Identifies the projection to create, where projection</td>
</tr>
</tbody>
</table>
forms to conventions described in Identifiers. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. The projection is created in the same schema as the anchor table.

**projection-column**

The name of a projection column. If you do not specify projection column names, Vertica uses the anchor table column names that are specified in the SELECT statement.

**grouped-clause**

See GROUPED Clause.

**ENCODING encoding-type**

Specifies the column encoding type, by default set to AUTO.

**ACCESSRANK integer**

Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking.

**table-col expr-with-table-cols**

A table column or expression of table columns to include in the projection.

**batch-udtf(batch-args)**

The batch UDTF to invoke each time the following events occur:

- Tuple mover mergeout
- Queries on the projection
- If invoked singly, on data load operations

**Important:** If the projection definition includes a pre-pass subquery, **batch-args** must exactly match the pre-pass UDTF output columns, in name and order.

**prepass-udtf (prepass-args)**

The pre-pass UDTF to invoke on each load operation such as COPY or INSERT.

If specified in a subquery, the pre-pass UDTF returns transformed data to the batch query for further processing. Otherwise, the pre-pass query results are added to projection data storage.
OVER (PARTITION BATCH BY partition-col-expr [, ...])

Specifies the UDTF type and how to partition the data it returns:

- BATCH identifies the UDTF as a batch UDTF.
- PREPASS identifies the UDTF as a pre-pass UDTF.

In both cases, the OVER clause specifies partitioning with one or more column expressions from the SELECT list. The first `partition-col-expr` is the first column expression in the SELECT list, the second `partition-col-expr` is the second column expression in the SELECT list, and so on.

**Note:** The projection is implicitly segmented and ordered on PARTITION BY columns.

<table>
<thead>
<tr>
<th><code>AS (batch-output-cols)</code></th>
<th>Optionally names columns that are returned by the UDTF. If a pre-pass subquery omits this clause, the outer batch query UDTF arguments (<code>batch-args</code>) must reference the column names as they are defined in the pre-pass UDTF.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AS (prepass-output-cols)</code></td>
<td>A schema table with the columns to include in the projection.</td>
</tr>
<tr>
<td><code>table-ref</code></td>
<td>Subquery result set that is returned to the outer batch UDTF.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser, or the following:

- Anchor table owner
- CREATE privilege on the schema
- EXECUTE privileges on all UDTFs that are referenced by the projection
Restrictions

Vertica does not regard live aggregate projections as superprojections, even one that includes all table columns.

UDTF Types

CREATE PROJECTION can define live aggregate projections that invoke user-defined transform functions (UDTFs). Vertica invokes UDFTs at multiple points of projection processing:

- **Pre-pass UDFTs:** Invoked when data is loaded into the projection's anchor table—for example through COPY or INSERT statements. A pre-pass UDTF transforms the new data before it is stored in the projection's ROS containers. You identify a pre-pass UDTF in the projection's PARTITION BY clause, through the keyword PREPASS.

- **Batch UDFTs:** Invoked on three events: the Vertica tuple mover consolidates stored projection data (mergeout); the projection is queried; and if invoked singly, data load operations. In all cases, the UDTF aggregates projection data and stores the aggregated results. Aggregation is cumulative across mergeout and load operations, and is completed (if necessary) on query execution. You identify a batch UDTF in the projection's PARTITION BY clause, through the keyword BATCH.

Vertica stores all UDTF results in projection ROS containers, thereby enabling faster response time when you query the projection.

UDTF Specification Options

A projection definition can specify up to two UDFTs, in any of the following ways:

- **Single pre-pass UDTF:** The pre-pass UDTF transforms newly loaded data and stores it in the projection. Use the following syntax:

  ```sql
  => CREATE PROJECTION projection-name AS SELECT ..., udtf(args)
     OVER(PARTITION PREPASS BY partition-cols) AS (prepass-output-columns) FROM table-ref;
  ```

- **Single batch UDTF:** The batch UDTF transforms and aggregates projection data on mergeout, insert, and query operations. Use the following syntax:
CREATE PROJECTION projection-name AS SELECT ..., udtf(args)
    OVER(PARTITION BATCH BY partition-cols) AS (batch-output-columns) FROM table-ref;

- **Pre-pass and batch UDTFs**: You can define a projection with a subquery that invokes a pre-pass UDTF. The pre-pass UDTF returns transformed data to the top-level batch query. The batch UDTF then iteratively aggregates all projection data.

Use the following syntax:

CREATE PROJECTION projection-name AS SELECT ..., batch-udtf(batch-args)
    OVER ( PARTITION BATCH BY partition-cols ) AS (batch-output-columns)
FROM ( SELECT ..., prepass-udtf(prepass-args)
    OVER ( PARTITION PREPASS BY partition-cols ) AS (prepass-output-columns)
    FROM table-ref ) sq-ref;

**Examples**

See Examples in Pre-Aggregating UDTF Results.

**See Also**

Pre-Aggregating UDTF Results

### CREATE RESOURCE POOL

Creates a custom resource pool and sets one or more resource pool parameters.

**Syntax**

CREATE RESOURCE POOL pool-name [ parameter-name setting ]...

**Parameters**

**Note**: You can set all resource pool parameters to their DEFAULT value. The V_CATALOG.RESOURCE_POOL_DEFAULTS system table contains default parameter values.
Query this table to determine default settings for all resource pools.

Default values specified in this table pertain only to user-defined resource pools. For built-in pool default values, see Built-In Pool Configuration.

<table>
<thead>
<tr>
<th>pool-name</th>
<th>The name of the resource pool. Built-in pool names cannot be used for user-defined pools.</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter-name</td>
<td>The parameter to set, listed below.</td>
</tr>
<tr>
<td>CASCADE TO</td>
<td>Specifies a secondary resource pool for executing queries that exceed the RUNTIMECAP setting of their assigned resource pool:</td>
</tr>
<tr>
<td></td>
<td>CASCADE TO secondary-pool</td>
</tr>
<tr>
<td>CPUAFFINITYMODE</td>
<td>Specifies whether the resource pool has exclusive or shared use of the CPUs specified in CPUAFFINITYSET:</td>
</tr>
<tr>
<td></td>
<td>CPUAFFINITYMODE {</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>• SHARED: Queries that run in this pool share its CPUAFFINITYSET CPUs with other Vertica resource pools.</td>
</tr>
<tr>
<td></td>
<td>• EXCLUSIVE: Dedicates CPUAFFINITYSET CPUs to this resource pool only, and excludes other Vertica resource pools. If CPUAFFINITYSET is set as a percentage, then that percentage of CPU resources available to Vertica is assigned solely for this resource pool.</td>
</tr>
<tr>
<td></td>
<td>• ANY (default): Queries in this resource pool can run on any CPU, invalid if CPUAFFINITYSET designates CPU resources.</td>
</tr>
<tr>
<td>CPUAFFINITYSET</td>
<td>Specifies which CPUs are available to this resource pool. All cluster nodes must have the same number of CPUs. The CPU resources assigned to this set are unavailable to general resource pools.</td>
</tr>
<tr>
<td></td>
<td>CPUAFFINITYSET {</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>cpu-index[, ...]: Dedicates one or more comma-delimited CPUs to this pool.</td>
<td></td>
</tr>
<tr>
<td>cpu-index_i-cpu-index_n: Dedications a range of contiguous CPU indexes to this pool</td>
<td></td>
</tr>
<tr>
<td>integer%: Percentage of all available CPUs to use for this pool. Vertica rounds this percentage down to include whole CPU units.</td>
<td></td>
</tr>
<tr>
<td>NONE (default): No affinity set is assigned to this resource pool. The queries associated with this pool are executed on any CPU.</td>
<td></td>
</tr>
</tbody>
</table>

**EXECUTIONPARALLELISM**

Limits the number of threads used to process any single query issued in this resource pool.

EXECUTIONPARALLELISM { integer | AUTO }

- **integer**: A value between 1 and the number of cores. Setting this parameter to a reduced value increases throughput of short queries issued in the pool, especially if the queries are executed concurrently.

- **AUTO** (default): Vertica sets this value based on the number of cores, available memory, and amount of data in the system. Unless memory is limited, or the amount of data is very small, Vertica sets this value to the number of cores on the node.

**MAXCONCURRENcy**

Sets the maximum number of concurrent execution slots available to the resource pool, across the cluster:

MAXCONCURRENcy { integer | NONE }

NONE (default) specifies unlimited number of concurrent execution slots.

**MAXMEMORYSIZE**

The maximum size per node the resource pool can grow by borrowing memory from the GENERAL pool:

MAXMEMORYSIZE { 'integer%' }
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>The amount of total memory available to the Vertica resource manager that is allocated to this pool per node:</td>
</tr>
<tr>
<td></td>
<td>MEMORYSIZE {</td>
</tr>
<tr>
<td></td>
<td>'integer%'</td>
</tr>
<tr>
<td></td>
<td>'integer{K</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>• <em>integer%</em>: Percentage of total memory</td>
</tr>
<tr>
<td></td>
<td>• *integer{K</td>
</tr>
<tr>
<td></td>
<td>• NONE: Unlimited; pool can borrow any amount of available memory from the GENERAL pool.</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>Specifies the preferred number queries to execute concurrently in the resource pool. This setting applies to the entire cluster:</td>
</tr>
<tr>
<td></td>
<td>PLANNEDCONCURRENCY { <em>integer</em></td>
</tr>
<tr>
<td></td>
<td>• <em>integer</em>: The preferred number of concurrently executing queries. When possible, query resource budgets are limited to allow this level of concurrent execution.</td>
</tr>
<tr>
<td></td>
<td>• AUTO (default): Value is calculated automatically at query runtime. Vertica sets this parameter to the lower of these two calculations, but never less than 4:</td>
</tr>
<tr>
<td></td>
<td>■ Number of logical cores</td>
</tr>
<tr>
<td></td>
<td>■ Memory divided by 2GB</td>
</tr>
</tbody>
</table>
For clusters where the number of logical cores differs on different nodes, AUTO can apply differently on each node. Distributed queries run like the minimal effective planned concurrency. Single node queries run with the planned concurrency of the initiator.

**Tip:** Change this parameter only after evaluating performance over a period of time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRIORITY</strong></td>
<td>Specifies priority of queries in this pool when they compete for resources in the GENERAL pool:</td>
</tr>
<tr>
<td></td>
<td>`PRIORITY { integer</td>
</tr>
<tr>
<td></td>
<td>• <code>integer</code>: A negative or positive integer value, where higher numbers denote higher priority:</td>
</tr>
<tr>
<td></td>
<td>- User-defined pools: -100 to 100</td>
</tr>
<tr>
<td></td>
<td>- Built-in pools SYSQUERY, RECOVERY, and TM: -110 to 110</td>
</tr>
<tr>
<td></td>
<td>• <code>HOLD</code>: Sets priority to -999. Queries in this pool are queued until QUEUETIMEOUT is reached.</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: 0</td>
</tr>
<tr>
<td><strong>QUEUETIMEOUT</strong></td>
<td>Species how long a request can wait for pool resources before it is rejected:</td>
</tr>
<tr>
<td></td>
<td>`QUEUETIMEOUT { integer</td>
</tr>
<tr>
<td></td>
<td>• <code>integer</code>: Maximum wait time in seconds</td>
</tr>
<tr>
<td></td>
<td>• <code>NONE</code>: No maximum wait time, request can be queued indefinitely.</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: 300 seconds</td>
</tr>
<tr>
<td><strong>RUNTIMECAP</strong></td>
<td>Prevents runaway queries by setting the maximum time a query in the pool can execute. If a query exceeds this setting, it tries to cascade to a secondary pool:</td>
</tr>
<tr>
<td></td>
<td>`RUNTIMECAP { interval</td>
</tr>
<tr>
<td></td>
<td>• <code>interval</code>: An interval of 1 minute or 100 seconds; should not exceed one year.</td>
</tr>
</tbody>
</table>
• NONE: No time limit on queries running in this pool.

   To specify a value in days, provide an integer value. To provide a value less than one day, provide the interval in the format hours:minutes:seconds. For example a value of 1:30:00 would equal 90 minutes.

   If the user or session also has a RUNTIMECAP, the shorter limit applies.

<table>
<thead>
<tr>
<th>RUNTIMEPRIORITYTHRESHOLD</th>
<th>Specifies in seconds a time limit in which a query must finish before the resource manager assigns to it the resource pool's RUNTIMEPRIORITY. All queries begin running at a HIGH priority. When a query's duration exceeds this threshold, it is assigned the RUNTIMEPRIORITY of the resource pool.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RUNTIMEPRIORITYTHRESHOLD seconds</td>
</tr>
<tr>
<td></td>
<td>Default: 2</td>
</tr>
</tbody>
</table>

Privileges
Superuser

Examples
This example shows how to create a resource pool with MEMORYSIZE of 1800 MB.

```sql
=> CREATE RESOURCE POOL ceo_pool MEMORYSIZE '1800M' PRIORITY 10;
CREATE RESOURCE POOL
```

Assuming the CEO report user already exists, associate this user with the preceding resource pool using ALTER USER statement.
GRANT USAGE ON RESOURCE POOL ceo_pool to ceo_user;
GRANT PRIVILEGE ALTER USER ceo_user RESOURCE POOL ceo_pool;
ALTER USER ceo_user RESOURCE POOL ceo_pool;

Issue the following command to confirm that the ceo_user is associated with the ceo_pool:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';
```

<table>
<thead>
<tr>
<th>user_id</th>
<th>45035996273733402</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>ceo_user</td>
</tr>
<tr>
<td>is_super_user</td>
<td>f</td>
</tr>
<tr>
<td>profile_name</td>
<td>default</td>
</tr>
<tr>
<td>is_locked</td>
<td>f</td>
</tr>
<tr>
<td>lock_time</td>
<td></td>
</tr>
<tr>
<td>resource_pool</td>
<td>ceo_pool</td>
</tr>
<tr>
<td>memory_cap_kb</td>
<td>unlimited</td>
</tr>
<tr>
<td>temp_space_cap_kb</td>
<td>unlimited</td>
</tr>
<tr>
<td>run_time_cap</td>
<td>unlimited</td>
</tr>
<tr>
<td>all_roles</td>
<td></td>
</tr>
<tr>
<td>default_roles</td>
<td>&quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
</tbody>
</table>

This example shows how to create and designate secondary resource pools.

```
=> CREATE RESOURCE POOL rp3 RUNTIMECAP '5 minutes';
=> CREATE RESOURCE POOL rp2 RUNTIMECAP '3 minutes' CASCADE TO rp3;
=> CREATE RESOURCE POOL rp1 RUNTIMECAP '1 minute' CASCADE TO rp2;
=> SET SESSION RESOURCE_POOL = rp1;
```

See Also

- ALTER RESOURCE POOL
- CREATE USER
- DROP RESOURCE POOL
- SET SESSION RESOURCE_POOL
- SET SESSION MEMORYCAP
- Managing Workloads

**Built-In Pools**

Vertica is preconfigured with built-in pools for various system tasks:
Built-in pools can be customized to suit your usage requirements. See `ALTER RESOURCE POOL` for details on resource pool settings.

**GENERAL**

A special, catch-all pool used to answer requests that have no specific resource pool associated with them. Any memory left over after memory has been allocated to all other pools is automatically allocated to the GENERAL pool. The `MEMORYSIZE` parameter of the GENERAL pool is undefined (variable), however, the GENERAL pool must be at least 1GB in size and cannot be smaller than 25% of the memory in the system.

The `MAXMEMORYSIZE` parameter of the GENERAL pool has special meaning; when set as a % value it represents the percent of total physical RAM on the machine that the Resource Manager can use for queries. By default, it is set to 95%. The GENERAL.MAXMEMORYSIZE governs the total amount of RAM that the Resource Manager can use for queries, regardless of whether it is set to a percent or to a specific value (for example, '10GB')

User-defined pools can borrow memory from the GENERAL pool to satisfy requests that need extra memory until the MAXMEMORYSIZE parameter of that pool is reached. If the pool is configured to have MEMORYSIZE equal to MAXMEMORYSIZE, it cannot borrow any memory from the GENERAL pool and is said to be a standalone resource pool. When multiple pools request memory from the GENERAL pool, they are granted access to general pool memory.
according to their priority setting. In this manner, the GENERAL pool provides some elasticity to account for point-in-time deviations from normal usage of individual resource pools.

Vertica recommends reducing the GENERAL pool MAXMEMORYSIZE if your catalog uses over 5% of overall memory.

**BLOBDATA**

The BLOBDATA pool controls resource usage for in-memory blobs. *In-memory blobs* are objects used by a number of the machine learning SQL functions. You should adjust this pool if you plan on processing large machine learning workloads. For information about tuning the pool, see Tuning for Machine Learning.

If a query using the BLOBDATA pool exceeds its query planning budget, then it spills to disk. For more information about tuning your query budget, see Target Memory Determination for Queries in Concurrent Environments.

**DBD**

The DBD pool controls resource usage for Database Designer processing. Use of this pool is enabled by configuration parameter `DBDUseOnlyDesignerResourcePool`, by default set to false.

By default, `QUEUETIMEOUT` is set to 0 for this pool. When resources are under pressure, this setting causes the DBD to time out immediately, and not be queued to run later. Database Designer then requests the user to run the designer later, when resources are more available.

![Important: Do not change QUEUETIMEOUT or other DBD pool settings.](image)

**JVM**

The JVM pool controls Java Virtual Machine resources used by Java User Defined Extensions. When a Java UDx starts the JVM, it draws resources from the those specified in the JVM resource pool. Vertica does not reserve memory in advance for the JVM pool. When needed, the pool can expand to 10% of physical memory or 2 GB of memory, whichever is smaller. If you are buffering large amounts of data, you may need to increase the size of the JVM resource pool.

You can adjust the size of your JVM resource pool by changing its configuration settings. Unlike other resource pools, the JVM resource pool does not release resources until a session is closed.
METADATA

The pool that tracks memory allocated for catalog data and storage data structures. This pool increases in size as Vertica metadata consumes additional resources. Memory assigned to the METADATA pool is subtracted from the GENERAL pool, enabling the Vertica resource manager to make more effective use of available resources. If the METADATA resource pool reaches 75% of the GENERAL pool, Vertica stops updating METADATA memory size and displays a warning message in the vertica.log file. You can enable or disable the METADATA pool with the EnableMetadataMemoryTracking general parameter.

If you have created a "dummy" or "swap" resource pool to protect resources for use by your operating system, you can replace that pool with the METADATA pool.

Users cannot change the parameters of the METADATA resource pool.

RECOVERY

The pool used by queries issued when recovering another node of the database. The MAXCONCURRENCY parameter is used to determine how many concurrent recovery threads to use. You can use the PLANNEDCONCURRENCY parameter (by default, set to twice the MAXCONCURRENCY) to tune how to apportion memory to recovery queries.

See Tuning for Recovery in the Administrator's Guide.

REFRESH

The pool used by queries issued by PROJECTION_REFRESHES operations. Refresh does not currently use multiple concurrent threads; thus, changes to the MAXCONCURRENCY values have no effect.

See Scenario: Tuning for Refresh in the Administrator's Guide.

SYSDATA

The pool reserved for temporary storage of intermediate results of queries against system monitoring and catalog tables. If the SYSDATA pool size is too low, Vertica cannot execute queries for large system tables or during high concurrent access to system tables.

Note: MAXMEMORYSIZE of the SYSDATA pool cannot be changed if any of its memory is in use.
SYSQUERY

The pool that runs queries against system monitoring and catalog tables. The SYSQUERY pool reserves resources for system table queries so that they are never blocked by contention for available resources.

TM

The Tuple Mover (TM) pool. You can set the MAXCONCURRENCY parameter for the TM pool to allow more than one concurrent TM operation to occur.

See Tuning Tuple Mover Pool Settings in the Administrator's Guide.

WOSDATA

The Write Optimized Store (WOS) resource pool. Data loads to the WOS automatically spill to the ROS once it exceeds a certain amount of WOS usage; the PLANNEDCONCURRENCY parameter of the WOS is used to determine this spill threshold. For instance, if PLANNEDCONCURRENCY of the WOSDATA pool is set to 4, once a load has occupied one quarter of the WOS, it spills to the ROS.

The WOSDATA pool is limited to a size of 2GB or 25% of the GENERAL pool's limits, whichever is less. However, when the GENERAL pool has a limit of 20GB or higher, the WOSDATA pool will instead have 2GB of dedicated memory.

Note: MAXMEMORYSIZE of the WOSDATA pool cannot be changed if any of its memory is in use. For example, you cannot change MAXMEMORYSIZE unless you first disable trickle loading jobs and wait until the WOS is empty.

See Scenario: Tuning for Continuous Load and Query in the Administrator's Guide.

Built-In Pool Configuration

The tables in this section list the default configuration settings for the Vertica built-in resource pools:

- GENERAL
- BLOBDATA
Some built-in resource pool parameter values have restrictions, which are noted in the tables. For guidance about how to tune the General resource pool, see *Best Practices for Managing Workload Resources* in the Administrator's Guide.

## GENERAL

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>N/A (cannot be set)</td>
</tr>
</tbody>
</table>
| MAXMEMORYSIZE  | Sets the maximum memory to use for all resource pools. Specify this value as a percentage of total RAM. For example, if your node has 64GB of memory, setting MAXMEMORYSIZE to 50% allocates half of the available memory. The total memory for all resource pools cannot exceed 32GB. Setting this parameter to 100% generates warning messages that swapping could result. Changing the MAXMEMORYSIZE parameter has the following restrictions:  
  - Must be 1GB or greater.  
  - Cannot be less than 25% of total system RAM. |
<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setting</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>Default: 95%, with minimum of 25% (at least 1GB), maximum 100%</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>0</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>Medium</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>2</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>300</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
</tbody>
</table>
| PLANNEDCONCURRENCY          | An integer representing the number of concurrent queries you expect to run against the resource pool. When set to the default value of AUTO, Vertica automatically sets PLANNEDCONCURRENCY at query runtime, choosing the lower of these two values:  
  - Number of cores  
  - Memory/2GB  
  
The value 4 is the minimum value for PLANNEDCONCURRENCY.  
For systems with a large number of cores, the PLANNEDCONCURRENCY setting for the GENERAL pool defaults to a value that is too low. For this configuration, Vertica recommends adjusting to a value equaling the number of cores:  
  ALTER RESOURCE POOL general PLANNEDCONCURRENCY <#cores>;  
**Default**: AUTO |
| MAXCONCURRENCY              | Unlimited  
Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.                                                                 |
| SINGLEINITIATOR             | False. This parameter is included for backwards compatibility only. Do not change the value.                                           |
### BLOBDATA

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>10</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>2</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>N/A (cannot be set)</td>
</tr>
</tbody>
</table>

### DBD

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>0</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>0</td>
</tr>
</tbody>
</table>
### Setting and Value

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>See GENERAL</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>Unlimited</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>False. This parameter is included for backwards compatibility. Do not change the value.</td>
</tr>
</tbody>
</table>

### JVM

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>10% of memory or 2 GB, whichever is smaller</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>0</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>2</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>300</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>AUTO</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>FALSE. This parameter is included for backwards compatibility. Do not change the value.</td>
</tr>
</tbody>
</table>
**METADATA**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>FALSE. This parameter is included for backwards compatibility. Do not change the value.</td>
</tr>
<tr>
<td>CPUAFFINITYSET</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>CPUAFFINITYMODE</td>
<td>N/A (cannot be set)</td>
</tr>
</tbody>
</table>

**RECOVERY**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Restrictions: cannot set to &lt; 25%.</td>
<td></td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>107</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>MEDIUM</td>
</tr>
</tbody>
</table>
### Setting | Value
--- | ---
RUNTIMEPRIORITYTHRESHOLD | 60
QUEUETIMEOUT | 300
RUNTIMECAP | NONE
PLANNEDCONCURRENCY | Twice MAXCONCURRENCY
MAXCONCURRENCY | \((\text{# of cores} / 2) + 1\)
Restrictions: Cannot set to 0 or NONE (unlimited)
SINGLEINITIATOR | True. This parameter is included for backwards compatibility. Do not change the value.

### REFRESH

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>-10</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>60</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>300</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>4</td>
</tr>
</tbody>
</table>

**Note:** PLANNEDCONCURRENCY for the REFRESH pool is always set to 4 by default (the AUTO setting). For some of the resource pools other than REFRESH, PLANNEDCONCURRENCY is set according to a formula that considers your hardware.
### Setting

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>configuration. PLANNEDCONCURRENCY for REFRESH does not consider hardware as the default setting is always 4. You can change the default setting of PLANNEDCONCURRENCY with the ALTER RESOURCE POOL statement.</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>Unlimited</td>
</tr>
<tr>
<td></td>
<td>Restrictions: cannot set to 0</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>True. This parameter is included for backwards compatibility. Do not change the value.</td>
</tr>
</tbody>
</table>

### SYSDATA

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>100m</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>Restriction: Setting To &lt;4m generates warnings that no system queries may be able to run in the system.</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>N/A (cannot be set)</td>
</tr>
</tbody>
</table>
## SYSQUERY

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>64M</td>
<td>Restrictions: Setting to &lt;20M generates warnings because it could prevent system queries from running and make problem diagnosis difficult.</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
<td></td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE.</td>
<td></td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>See GENERAL</td>
<td></td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>Unlimited</td>
<td>Restrictions: Setting to 0 generates warnings that no system queries may be able to run in the system.</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>False. This parameter is included for backwards compatibility only. Do not change the value.</td>
<td></td>
</tr>
</tbody>
</table>

## TM

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>100M</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>Unlimited</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>AUTO</td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>105</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>60</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>300</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>1</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Restrictions: Cannot set to 0 or NONE (unlimited)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>True. This parameter is included for backwards compatibility. Do not change the value.</td>
</tr>
</tbody>
</table>

**WOSDATA**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORYSIZE</td>
<td>0%</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>25% or 2GB of the GENERAL pool's limits, whichever is less. When the GENERAL pool's memory limits are 20GB or greater, the WOSDATA pool will instead have a dedicated 2GB of memory.</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMEPRIORITYTHRESHOLD</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>QUEUETIMEOUT</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>NONE</td>
</tr>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>2</td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>N/A (cannot be set)</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>N/A (cannot be set)</td>
</tr>
</tbody>
</table>

## CREATE ROLE

Creates a role. After creating a role, use **GRANT statements** to specify role permissions.

### Syntax

```
CREATE ROLE role
```

### Parameters

| role | The name for the new role, where `role` conforms to conventions described in **Identifiers**. |

### Privileges

Superuser

### Examples

This example shows to create an empty role called `roleA`.

```sql
=> CREATE ROLE roleA;
CREATE ROLE
```
See Also

- ALTER ROLE RENAME
- DROP ROLE

**CREATE SCHEMA**

Defines a schema.

**Syntax**

```sql
CREATE SCHEMA [ IF NOT EXISTS ] [ database. ] schema
... [ AUTHORIZATION username]
... [ DEFAULT { INCLUDE | EXCLUDE } [ SCHEMA ] PRIVILEGES ]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF NOT EXISTS</td>
<td>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created. The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does. For related information, see ON_ERROR_STOP.</td>
</tr>
<tr>
<td>[ database. ] schema</td>
<td>Identifies the schema to create, where schema conforms to conventions described in Identifiers. This name must be unique among all other schema names in the database. If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>AUTHORIZATION username</td>
<td>Assigns ownership of the schema to a user. If a user name is not provided, the user who creates the schema is assigned ownership. Only superusers can create a schema that is</td>
</tr>
</tbody>
</table>
DEFAULT
{INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES

Specifies whether to enable or disable inheritance of privileges for tables in the schema. INCLUDE PRIVILEGES grants the tables in the schema the same privileges granted to the schema.

If you omit INCLUDE PRIVILEGES, you must grant privileges individually for each table in the schema.

For more information see Grant Inherited Privileges.

### Privileges

- Superuser
- **CREATE privilege for the database**

Optionally, **CREATE SCHEMA** can include the following sub-statements to create tables within the schema:

- **CREATE TABLE**
- **GRANT Statements**

These sub-statements are treated as if they were entered as individual commands after **CREATE SCHEMA** executes. The following exceptions apply:

- The AUTHORIZATION statement indicates all tables are owned by the specified user.
- **CREATE SCHEMA** statement and all associated sub-statements are treated as a single transaction. If any statement fails, Vertica rolls back the entire **CREATE SCHEMA** statement.

### Examples

The following example creates a schema named s1 with no objects.

```
=> CREATE SCHEMA s1;
```

The following command creates schema s2 if it does not already exist:

```
=> CREATE SCHEMA IF NOT EXISTS schema2;
```
If the schema already exists, Vertica returns a rollback message:

```sql
=> CREATE SCHEMA IF NOT EXISTS schema2;
NOTICE 4214: Object "schema2" already exists; nothing was done
```

The following series of commands create a schema named s1 with a table named t1 and grants Fred and Aniket access to all existing tables and ALL privileges on table t1:

```sql
=> CREATE SCHEMA s1;
=> CREATE TABLE t1 (c INT);
=> GRANT USAGE ON SCHEMA s1 TO Fred, Aniket;
=> GRANT ALL ON TABLE t1 TO Fred, Aniket;
```

This example sets the default behavior for new table t2 to automatically inherit the schema's privileges:

```sql
=> CREATE SCHEMA s1 DEFAULT INCLUDE SCHEMA PRIVILEGES;
=> CREATE TABLE s1.t2(i INT);
```

This example sets the default for new tables to not automatically inherit privileges from the schema:

```sql
=> CREATE SCHEMA s2 DEFAULT EXCLUDE SCHEMA PRIVILEGES;
```

See Also

- ALTER SCHEMA
- SET SEARCH_PATH
- DROP SCHEMA

**CREATE SEQUENCE**

Defines a new named sequence number generator object. Along with AUTO_INCREMENT and IDENTITY sequences, you can use named sequences to set the default values of primary key columns. Sequences guarantee uniqueness, and avoid constraint enforcement problems and overhead. For more information about sequence types and their usage, see Sequence Types in the Administrator's Guide.
Syntax

CREATE SEQUENCE [[\textit{database}\textperiodcentered}\textit{schema}\textperiodcentered]\textit{sequence} ...
[ \textit{INCREMENT} \text{BY} \ \textit{integer} ] ...
[ MINVALUE \ \textit{integer} \ | \ \text{NO} \ MINVALUE ] ...
[ MAXVALUE \ \textit{maxvalue} \ | \ \text{NO} \ MAXVALUE ] ...
[ \text{START} \ \text{WITH} \ \textit{integer} ] ...
[ \textit{CACHE} \ | \ \text{NO} \ \text{CACHE} ] ...
[ CYCLE \ | \ \text{NO} \ CYCLE ]

Parameters

<table>
<thead>
<tr>
<th>[\textit{database}\textperiodcentered]\textit{schema}</th>
<th>\textbf{Specifies a schema}, by default public. If \textit{schema} is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{myschema.thisDbObject}</td>
</tr>
<tr>
<td>If you specify a database, it must be the current database.</td>
<td></td>
</tr>
<tr>
<td>\textit{sequence}</td>
<td>Identifies the sequence to create, where \textit{sequence-name} conforms to conventions described in \texttt{Identifiers}. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.</td>
</tr>
<tr>
<td>\texttt{INCREMENT [BY] } \textit{integer}</td>
<td>A positive or negative integer that specifies how much to increment or decrement the sequence on each call to \texttt{NEXTVAL}, by default set to 1.</td>
</tr>
<tr>
<td>\texttt{Note: Setting this parameter to } \textit{integer} \texttt{guarantees that column values always increment by at least } \textit{integer}. However, column values can sometimes increment by more than \textit{integer} unless you also set the \text{NO} \ \text{CACHE} parameter.</td>
<td></td>
</tr>
<tr>
<td>\texttt{MINVALUE } \textit{integer} \text{NO MINVALUE (default)}</td>
<td>Determines the minimum value a sequence can generate. If you omit this clause or specify NO MINVALUE, default values are used: 1 and \text{2}^{63}-1 for ascending and descending sequences, respectively.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{MAXVALUE } \textit{integer} \text{NO MAXVALUE (default)}</td>
<td>Determines the maximum value for the sequence. If you omit this clause or specify NO MAXVALUE, default values are used: \text{2}^{63}-1 and -1 for ascending and descending sequences, respectively.</td>
</tr>
<tr>
<td>Clause</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>START [WITH] integer</td>
<td>Sets the sequence start value to <code>integer</code>. The next call to <code>NEXTVAL</code> returns <code>integer</code>. If you omit this clause, the sequence start value is set to MINVALUE for ascending sequences, and MAXVALUE for descending sequences.</td>
</tr>
</tbody>
</table>
| CACHE integer NO CACHE | Specifies whether to cache unique sequence numbers on each node for faster access. `CACHE` takes an integer argument as follows:  
  - >1 specifies how many unique numbers each node caches per session.  
  - 0 or 1 specifies to disable caching (equivalent to NO CACHE).  
If you omit this clause, the sequence cache is set to 250,000. For details, see Sequence Caching in the Administrator's Guide. |
| CYCLE NO CYCLE (default) | Specifies whether the sequence can wrap when its minimum or maximum values are reached:  
  - CYCLE: The sequence wraps as follows:  
    - When an incrementing sequence reaches its upper limit, it is reset to its minimum value.  
    - When a decrementing sequence reaches its lower limit, it is reset to its maximum value.  
  - NO CYCLE (default): Calls to `NEXTVAL` return an error after the sequence reaches its maximum or minimum value. |

**Privileges**

Superuser, or user with CREATE privilege on the sequence schema.

For details on privileges required to use and manage sequence, see Named Sequence Privileges in the Administrator's Guide.

**Examples**

See Also

- AUTO_INCREMENT and IDENTITY Sequences in the Administrator's Guide
- ALTER SEQUENCE
- DROP SEQUENCE

**CREATE SUBNET**

Identifies the subnet to which the nodes of a Vertica database belong. Use this statement to configure import/export from a database to other Vertica clusters.

**Syntax**

```
CREATE SUBNET subnet-name WITH 'subnet-prefix'
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>subnet-name</code></td>
<td>A name you assign to the subnet, where <code>subnet-name</code> conforms to conventions described in Identifiers.</td>
</tr>
<tr>
<td><code>subnet-prefix</code></td>
<td>The routing prefix expressed in quad-dotted decimal representation. Refer to system table NETWORK_INTERFACES to get the prefix of all available IP networks.</td>
</tr>
</tbody>
</table>

You can then configure the database to use the subnet for import/export. For details, see Identify the Database or Nodes Used for Import/Export in the Administrator's Guide.

**Privileges**

dbadmin
Examples

=> CREATE SUBNET mySubnet WITH '123.4.5.6';

**CREATE TABLE**

Creates a table in the logical schema.

**Syntax**

**Create with column definitions**

```
CREATE TABLE [ IF NOT EXISTS ] [ [[database.]schema.]table
... ( column-definition[, ... ] )
... [ table-constraint ]
... [ load-method ]
... [ ORDER BY column[, ... ] ]
... [ segmentation-spec ]
... [ KSAFE [ k-num ] ]
... [ partition-clause ]
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

**Create from another table**

```
CREATE TABLE [ IF NOT EXISTS ] [ [[database.]schema.]table ( AS-clause | LIKE-clause )
```

**AS-clause**

```
... [ ( column-name-list ) ]
... [ load-method ]
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list ]
```

**LIKE-clause**

```
LIKE [[database.]schema.]existing-table
...[ {INCLUDING | EXCLUDING} PROJECTIONS ]
...[ load-method ]
...[ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

**Parameters**

| **IF NOT EXISTS** | Specifies to generate an informational message if an object |
already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.

The IF NOT EXISTS clause is useful for SQL scripts where you want to create an object if it does not already exist, and reuse the existing object if it does.

For related information, see **ON_ERROR_STOP**.

<table>
<thead>
<tr>
<th><strong>[database.]schema</strong></th>
<th>Specifies a schema, by default public. If <em>schema</em> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

| **table** | Identifies the table to create, where *table* conforms to conventions described in **Identifiers**. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema. |

| **column-definition** | Defines a table column. A table can have up to 1600 columns.                                           |

| **table-constraint**  | Adds a constraint to table metadata.                                                                      |

<table>
<thead>
<tr>
<th><strong>load-method</strong></th>
<th>Specifies default load behavior for all DML operations on this table, such as INSERT and COPY, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- AUTO (default): Initially loads data into WOS, suitable for smaller bulk loads.</td>
</tr>
<tr>
<td></td>
<td>- DIRECT: Loads data directly into ROS containers, suitable for large (&gt;100 MB) bulk loads.</td>
</tr>
<tr>
<td></td>
<td>- TRICKLE: Loads data only into WOS, suitable for frequent incremental loads.</td>
</tr>
</tbody>
</table>

For details, see **Choosing a Load Method** in the Administrator's Guide.

| **ORDER BY column [,...]** | Specifies columns from the SELECT list on which to sort the superprojection that is automatically created for this table. The ORDER BY clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order. |
If you omit the ORDER BY clause, Vertica uses the SELECT list order as the projection sort order. This option is invalid for external tables.

### segmentation-spec

**Invalid for external tables**, specifies how to distribute data for auto-projections of this table. Supply one of the following clauses:

- **hash-segmentation-clause**: Specifies to segment data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables. For details, see Hash Segmentation Clause.

- **unsegmented-clause**: Specifies to create an unsegmented projection. For details, see Unsegmented Clause.

If this clause is omitted, Vertica generates auto-projections with default hash segmentation.

### KSAFE \([k\text{-num}]\)

Specifies K-safety of auto-projections created for this table, where \(k\text{-num}\) must be equal to or greater than system K-safety. If you omit this option, the projection uses the system K-safety level. For general information, see K-Safety in Vertica Concepts.

**Note:** This option is invalid for external tables.

### partition-clause

Logically divides table data storage through a PARTITION BY clause:

```
PARTITION BY partition-expression [ GROUP BY group-expression ]
```

For details, see Partition Clause.

This option is invalid for external tables.

### column-name-list

Valid only when creating a table from a query (AS **query**), defines column names that map to the query output. If you omit this list, Vertica uses the query output column names. The names in **column-name-list** and queried columns must be the same in number.

For example:

```
CREATE TABLE customer_occupations (name, profession)
AS SELECT customer_name, occupation FROM customer_dimension;
```
This clause and the ENCODED BY clause are mutually exclusive. Column name lists are invalid for external tables.

### { INCLUDE | EXCLUDE } [SCHEMA] PRIVILEGES

Specifies default inheritance of schema privileges for this table:

- **EXCLUDE [SCHEMA] PRIVILEGES** (default) disables inheritance of privileges from the schema
- **INCLUDE [SCHEMA] PRIVILEGES** grants the table the same privileges granted to its schema

For more information see [Grant Inherited Privileges](#).

### AS *query*

Creates and loads a table from the results of a query, specified as follows:

```sql
AS [ /*+hint[, hint]*/ ] [ AT epoch ] query
```

You can qualify the AS clause with one or both of the following hints:

- A load method hint: **AUTO, DIRECT, or TRICKLE**
  
  **Note:** The CREATE TABLE statement can also specify a load method. However, this load method applies to load operations only after the table is created.

- **LABEL**

  For details, see [Creating a Table from a Query](#) in the Administrator's Guide.

### ENCODED BY *column-ref-list*

A list of columns from the source table, where each column is qualified by one or both of the following encoding options:

- **ACCESSRANK integer**: Overrides the default access rank for a column, useful for prioritizing access to a column. See [Prioritizing Column Access Speed](#) in the Administrator's Guide.

- **ENCODING encoding-type**: Specifies the type of encoding to use on the column. The default encoding type is AUTO.

This option and *column-name-list* are mutually exclusive. This option is invalid for external tables.
LIKE `existing-table`

Creates the table by replicating an existing table. You can qualify the LIKE clause with one of the following options:

- **EXCLUDING PROJECTIONS** (default): Do not copy projections from the source table.
- **INCLUDING PROJECTIONS**: Copy current projections from the source table for the new table.
- **Load-method**: See description above.
- `{INCLUDE|EXCLUDE} [SCHEMA] PRIVILEGES`: See description above.

For details, see [Replicating a Table](https://www.vertica.com/support/documentation/9.0.x/manuals/sqlreference_manual/page3468.html) in the Administrator's Guide.

### Privileges

The following privileges are required:

- **CREATE** privileges on the table schema

- If creating a table that includes a named sequence:
  - **SELECT** privilege on sequence object
  - **USAGE** privilege on sequence schema

- If creating a table with the LIKE clause, owner privileges on the source table

### Examples

In the Administrator's Guide, see:

- [Creating Tables](https://www.vertica.com/support/documentation/9.0.x/manuals/sqlreference_manual/page3468.html)
- [Replicating a Table](https://www.vertica.com/support/documentation/9.0.x/manuals/sqlreference_manual/page3468.html)
- [Creating a Table from a Query](https://www.vertica.com/support/documentation/9.0.x/manuals/sqlreference_manual/page3468.html)
See Also

- CREATE TEMPORARY TABLE
- CREATE EXTERNAL TABLE AS COPY
- CREATE FLEX TABLE

Column-Definition

Specifies the name, data type, and constraints to be applied to a column.

Syntax

```
column-name  data-type
...  [ column-constraint ]
...  [ ENCODING  encoding-type ]
...  [ ACCESSRANK  integer ]
```

Parameters

<table>
<thead>
<tr>
<th>column-name</th>
<th>The name of a column to be created or added.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-type</td>
<td>One of the following data types:</td>
</tr>
<tr>
<td></td>
<td>• BINARY</td>
</tr>
<tr>
<td></td>
<td>• BOOLEAN</td>
</tr>
<tr>
<td></td>
<td>• CHARACTER</td>
</tr>
<tr>
<td></td>
<td>• DATE/TIME</td>
</tr>
<tr>
<td></td>
<td>• NUMERIC</td>
</tr>
</tbody>
</table>

For information on data types, see SQL Data Types.

Tip: When specifying the maximum column width in a CREATE TABLE statement, use the width in bytes (octets) for any of the string types. Each UTF-8 character might
require four bytes, but European languages generally require a little over one byte per character, while Oriental languages generally require a little under three bytes per character.

<table>
<thead>
<tr>
<th><strong>column_constraint</strong></th>
<th>Specifies a column constraint for this column.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENCODING encoding-type</strong></td>
<td>Specifies the column encoding type, by default set to AUTO.</td>
</tr>
<tr>
<td><strong>ACCESSRANK integer</strong></td>
<td>Overrides the default access rank for a column. Use this parameter to increase or decrease the speed at which Vertica accesses a column. For more information, see Overriding Default Column Ranking.</td>
</tr>
</tbody>
</table>

**Example**

The following example creates a table named Employee_Dimension and its associated superprojection in the Public schema. The Employee_key column is designated as a primary key, and RLE encoding is specified for the Employee_gender column definition:

```sql
=> CREATE TABLE Public.Employee_Dimension ( 
   Employee_key integer PRIMARY KEY NOT NULL, 
   Employee_gender varchar(8) ENCODING RLE, 
   Courtesy_title varchar(8), 
   Employee_first_name varchar(64), 
   Employee_middle_initial varchar(8), 
   Employee_last_name varchar(64), 
);
```

**Column-Name-List**

Used to rename columns when creating a table or temporary table from a query (CREATE TABLE AS SELECT); also used to specify the column's encoding type and access rank.

**Syntax**

```
column-name-list
... [ ENCODING encoding-type ]
... [ ACCESSRANK integer ]
... [ GROUPED ( column-reference[,....] ) ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column-name</td>
<td>Specifies the new name for the column.</td>
</tr>
<tr>
<td>ENCODING encoding-type</td>
<td>Specifies the type of encoding to use on the column. The default encoding type is AUTO.</td>
</tr>
<tr>
<td>ACCESSRANK integer</td>
<td>Overrides the default access rank for a column, useful for prioritizing access to a column. See Prioritizing Column Access Speed in the Administrator's Guide.</td>
</tr>
<tr>
<td>GROUPED</td>
<td>Groups two or more columns. For detailed information, see GROUPED Clause.</td>
</tr>
</tbody>
</table>

Requirements

- A column in the list can not specify the column's data type or any constraint. These are derived from the queried table.

- If the query output has expressions other than simple columns (for example, constants or functions) then an alias must be specified for that expression, or the column name list must include all queried columns.

- CREATE TABLE can specify encoding types and access ranks in the column name list or the query's ENCODED BY clause, but not in both. For example, the following CREATE TABLE statement sets encoding and access rank on two columns in the column name list:

  ```sql
  => CREATE TABLE promo1 (state ENCODING RLE ACCESSRANK 1, zip ENCODING RLE, ...)
      AS SELECT * FROM customer_dimension ORDER BY customer_state;
  ```

  The next statement specifies the same encoding and access rank in the query's ENCODED BY clause.

  ```sql
  => CREATE TABLE promo2
      AS SELECT * FROM customer_dimension ORDER BY customer_state
      ENCODED BY customer_state ENCODING RLE ACCESSRANK 1, customer_zip ENCODING RLE;
  ```
Column-Constraint

Adds a constraint to a column's metadata. See Adding Constraints in the Administrator's Guide.

Syntax

**Permanent table**

```sql
[ { AUTO_INCREMENT | IDENTITY } [ (args) ] ]
[ CONSTRAINT constraint-name ]
[ ...[ CHECK (expression) [ ENABLED | DISABLED ] ]
[ ...[ DEFAULT default-expr ] [ SET USING using-expr ] | DEFAULT USING expr ]
[ ...[ NULL | NOT NULL ]
[ ...[ { PRIMARY KEY | REFERENCES table [ (column) ] } [ ENABLED | DISABLED ] ]
[ ...[ UNIQUE [ ENABLED | DISABLED ] ]
```

**Temporary table**

```sql
[ CONSTRAINT constraint-name ]
[ ...[ CHECK (expression) [ ENABLED | DISABLED ] ]
[ ...[ DEFAULT default-expr ]
[ ...[ NULL | NOT NULL ]
[ ...{ PRIMARY KEY | REFERENCES table [ (column) ] } [ ENABLED | DISABLED ] ]
[ ...{ SET USING using-expr ]
[ ...{ UNIQUE [ ENABLED | DISABLED ] ]
```

Parameters

**Note:** A number of parameters can be qualified with the keyword ENABLED or DISABLED. For details, see Enforcing Constraints below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_INCREMENT IDENTITY</td>
<td>Creates a table column whose values are automatically generated by the database, and cannot be changed. You can set this constraint on only one table column. AUTO_INCREMENT and IDENTITY are synonyms. For details this constraint and optional arguments, see AUTO_INCREMENT and IDENTITY Sequences in the Administrator's Guide.</td>
</tr>
<tr>
<td>CONSTRAINT constraint-name</td>
<td>Assigns a name to the constraint. Vertica recommends that you name all constraints.</td>
</tr>
<tr>
<td>CHECK</td>
<td>Adds check condition expression, which returns a Boolean value.</td>
</tr>
<tr>
<td><strong>expression</strong></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>---</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Specifies this column's default value:</td>
</tr>
<tr>
<td></td>
<td><strong>DEFAULT default-exp</strong></td>
</tr>
<tr>
<td></td>
<td>Vertica evaluates the DEFAULT expression and sets the column on load operations, if the operation omits a value for the column. For details about valid expressions, see Defining Column Values.</td>
</tr>
<tr>
<td>SET USING</td>
<td>Specifies to set values in this column from the specified expression:</td>
</tr>
<tr>
<td></td>
<td><strong>SET USING using-exp</strong></td>
</tr>
<tr>
<td></td>
<td>Vertica evaluates the SET USING expression and refreshes column values only when the function REFRESH_COLUMNS is invoked. For details about valid expressions, see Defining Column Values.</td>
</tr>
<tr>
<td>DEFAULT USING</td>
<td>Defines the column with DEFAULT and SET USING constraints, specifying the same expression for both. DEFAULT USING columns support the same expressions as SET USING columns, and are subject to the same restrictions.</td>
</tr>
<tr>
<td>NULL NOT NULL</td>
<td>Specifies whether the column can contain null values:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NULL</strong>: Allows null values in the column. If you set this constraint on a primary key column, Vertica ignores it and sets it to NOT NULL.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NOT NULL</strong>: Specifies that the column must be set to a value during insert and update operations. If the column has no default value and no value is provided, INSERT or UPDATE returns an error.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you omit this constraint, the default is NULL for all columns except primary key columns, which Vertica always sets to NOT NULL.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>External tables</strong>: If you specify NOT NULL and the column contains null values, queries are liable to return errors or generate unexpected behavior. Specify NOT NULL for an external table column only if you are sure that the column does not contain nulls.</td>
</tr>
<tr>
<td><strong>PRIMARY KEY</strong></td>
<td>Defines the column as the table's primary key.</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>Identifies this column as a foreign key:</td>
</tr>
<tr>
<td></td>
<td>REFERENCES <em>table</em> [<em>column</em>]</td>
</tr>
<tr>
<td></td>
<td>where <em>column</em> is the primary key in <em>table</em>. If you omit <em>column</em>, Vertica references the primary key in <em>table</em>.</td>
</tr>
<tr>
<td><strong>UNIQUE</strong></td>
<td>Requires column data to be unique with respect to all table rows.</td>
</tr>
</tbody>
</table>

**Privileges**

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

**Enforcing Constraints**

The following constraints can be qualified with the keyword ENABLED or DISABLED:

- **PRIMARY KEY**
- **UNIQUE**
- **CHECK**

If you omit ENABLED or DISABLED, Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- EnableNewPrimaryKeysByDefault
- EnableNewUniqueKeysByDefault
- EnableNewCheckConstraintsByDefault

For details, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.
Partition Clause

Specifies partitioning of table data, through a PARTITION BY clause in the table definition:

```
PARTITION BY partition-expression [ GROUP BY group-expression ]
```

<table>
<thead>
<tr>
<th>PARTITION BY partition-expression</th>
<th>For each table row, resolves to a partition key that is derived from one or more table columns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP BY group-expression</td>
<td>For each table row, resolves to a partition group key that is derived from the partition key. Vertica uses group keys to merge partitions into separate partition groups. GROUP BY must use the same expression as PARTITION BY. For example:</td>
</tr>
<tr>
<td></td>
<td><code>...PARTITION BY (i+j) GROUP BY (CASE WHEN (i+j) &lt; 5 THEN 1 WHEN (i+j) &lt; 10 THEN 2 ELSE 3);</code></td>
</tr>
</tbody>
</table>

**Caution:** Avoid partitioning tables on LONG VARBINARY and LONG VARCHAR columns. Doing so can adversely impact performance.

For details on partitioning table data by groups, see Partition Grouping and Hierarchical Partitioning in the Administrator's Guide.

Partitioning Requirements and Restrictions

PARTITION BY expressions can specify leaf expressions, functions, and operators. The following requirements and restrictions apply:

- All table projections must include all columns referenced in the expression; otherwise, Vertica cannot resolve the expression.

- The expression can reference multiple columns, but it must resolve to a single non-null value for each row.

- All leaf expressions must be constants or table columns.
- All other expressions must be functions and operators. The following restrictions apply to functions:
  - They must be immutable—that is, they return the same value regardless of time and locale and other session- or environment-specific conditions.
  - They cannot be aggregate functions.
  - They cannot be Vertica meta-functions.
- The expression cannot include queries.

GROUP BY expressions do not support modulo (%) operations.

See Also

Partitioning Tables in the Administrator's Guide

Table-Constraint

Adds a constraint to table metadata. You can specify table constraints with CREATE TABLE, or add a constraint to an existing table with ALTER TABLE. For details, see Adding Constraints in the Administrator's Guide.

Note: Adding a constraint to a table that is referenced in a view does not affect the view.

Syntax

```
[ CONSTRAINT constraint-name ]
{
  ... PRIMARY KEY (column[,...]) [ ENABLED | DISABLED ]
  ... | FOREIGN KEY (column[,...]) REFERENCES table [ (column[,...]) ]
  ... | UNIQUE (column[,...]) [ ENABLED | DISABLED ]
  ... | CHECK (expression) [ ENABLED | DISABLED ]
}
```

Parameters

<p>| CONSTRAINT constraint-name | Assigns a name to the constraint. Vertica recommends that you name all constraints. |</p>
<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRIMARY KEY</strong></td>
<td>Defines one or more NOT NULL columns as the primary key as follows:</td>
<td>PRIMARY KEY (column[,...]) [ ENABLED</td>
</tr>
<tr>
<td></td>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you do not name a primary key constraint, Vertica assigns the name C_PRIMARY.</td>
<td></td>
</tr>
<tr>
<td><strong>FOREIGN KEY</strong></td>
<td>Adds a referential integrity constraint defining one or more columns as foreign keys as follows:</td>
<td>FOREIGN KEY (column[,...]) REFERENCES table [(column[,...])]</td>
</tr>
<tr>
<td></td>
<td>If you omit column references, If you omit column, Vertica references the primary key in table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you do not name a foreign key constraint, Vertica assigns the name C_FOREIGN.</td>
<td></td>
</tr>
<tr>
<td><strong>UNIQUE</strong></td>
<td>Specifies that the data in a column or group of columns is unique with respect to all table rows, as follows:</td>
<td>UNIQUE (column[,...]) [ENABLED</td>
</tr>
<tr>
<td></td>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you do not name a unique constraint, Vertica assigns the name C_UNIQUE.</td>
<td></td>
</tr>
<tr>
<td><strong>CHECK</strong></td>
<td>Specifies a check condition as an expression that returns a Boolean value, as follows:</td>
<td>CHECK (expression) [ENABLED</td>
</tr>
<tr>
<td></td>
<td>You can qualify this constraint with the keyword ENABLED or DISABLED. See Enforcing Constraints below.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you do not name a check constraint, Vertica assigns the name C_CHECK.</td>
<td></td>
</tr>
</tbody>
</table>
Privileges

Table owner or user WITH GRANT OPTION is grantor.

- REFERENCES privilege on table to create foreign key constraints that reference this table
- USAGE privilege on schema that contains the table

Enforcing Constraints

A table can specify whether Vertica automatically enforces a primary key, unique key or check constraint with the keyword ENABLED or DISABLED. If you omit ENABLED or DISABLED, Vertica determines whether to enable the constraint automatically by checking the appropriate configuration parameter:

- EnableNewPrimaryKeysByDefault
- EnableNewUniqueKeysByDefault
- EnableNewCheckConstraintsByDefault

For details, see Enforcing Primary Key, Unique Key, and Check Constraints Automatically.

Examples

The following example creates a table (t01) with a primary key constraint.

```
CREATE TABLE t01 (id int CONSTRAINT sampleconstraint PRIMARY KEY);
```

This example creates the same table without the constraint, and then adds the constraint with ALTER TABLE ADD CONSTRAINT

```
CREATE TABLE t01 (id int);
ALTER TABLE t01 ADD CONSTRAINT sampleconstraint PRIMARY KEY(id);
WARNING 2623: Column "id" definition changed to NOT NULL
```

The following example creates a table (addapk) with two columns, adds a third column to the table, and then adds a primary key constraint on the third column.

```
CREATE TABLE addapk (col1 INT, col2 INT);

CREATE TABLE

ALTER TABLE addapk ADD COLUMN col3 INT;

ALTER TABLE

ALTER TABLE addapk ADD CONSTRAINT col3constraint PRIMARY KEY (col3) ENABLED;

WARNING 2623: Column "col3" definition changed to NOT NULL

ALTER TABLE

Using the sample table addapk, check that the primary key constraint is enabled (is_enabled is t).

=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col3constraint</td>
<td>col3</td>
<td>p</td>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)

This example disables the constraint using ALTER TABLE ALTER CONSTRAINT.

=> ALTER TABLE addapk ALTER CONSTRAINT col3constraint DISABLED;

Check that the primary key is now disabled (is_enabled is f).

=> SELECT constraint_name, column_name, constraint_type, is_enabled FROM PRIMARY_KEYS WHERE table_name IN ('addapk');

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>column_name</th>
<th>constraint_type</th>
<th>is_enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>col3constraint</td>
<td>col3</td>
<td>p</td>
<td>f</td>
</tr>
</tbody>
</table>

(1 row)

For a general discussion of constraints, see About Constraints. For additional examples of creating and naming constraints, see Naming Constraints.

CREATE TEMPORARY TABLE

Creates a table whose data persists only during the current session. Temporary table data is not visible to other sessions.
Syntax

Create with column definitions

  ... ( column-definition[,... ] )
  ... [ table-constraint ]
  ... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
  ... [ Load-method ]
  ... [ NO PROJECTION ]
  ... [ ORDER BY table-column[,... ]]
  ... [ segmentation-spec ]
  ... [ KSAFE [k-num] ]
  ... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]

Create from another table

CREATE TEMP[ORARY] TABLE [ IF NOT EXISTS ] [[database.]schema.]table-name
  ... ( column-name-list )
  ... [ ON COMMIT { DELETE | PRESERVE } ROWS ]
  ... [ Load-method ]
AS [ /*+ hint[, hint] */ ] [ AT epoch ] query [ ENCODED BY column-ref-list ]

Parameters

<table>
<thead>
<tr>
<th>scope</th>
<th>Specifies visibility of the table definition:</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL (default): The table definition is visible to all sessions, and persists until you explicitly drop the table.</td>
<td></td>
</tr>
<tr>
<td>LOCAL: the table definition is visible only to the session in which it is created, and is dropped when the session ends.</td>
<td></td>
</tr>
</tbody>
</table>

Regardless of this setting, retention of temporary table data is set by the keywords ON COMMIT DELETE and ON COMMIT PRESERVE (see below).

For more information, see Creating Temporary Tables in the Administrator's Guide.

<table>
<thead>
<tr>
<th>IF NOT EXISTS</th>
<th>Specifies to generate an informational message if an object already exists under the specified name. If you omit this option and the object exists, Vertica generates a ROLLBACK error message. In both cases, the object is not created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The IF NOT EXISTS clause is useful for SQL scripts where you</td>
<td></td>
</tr>
</tbody>
</table>
want to create an object if it does not already exist, and reuse the existing object if it does.

For related information, see **ON_ERROR_STOP**.

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If <em>schema</em> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database. If you do not specify a schema, the table is created in the default schema.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table-name</th>
<th>Identifies the table to create, where <em>table-name</em> conforms to conventions described in <em>identifiers</em>. It must also be unique among all names of sequences, tables, projections, views, and models within the same schema.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>column-definition</th>
<th>Defines a table column. A table can have up to 1600 columns.</th>
<th>table-constraint</th>
<th>Adds a constraint to table metadata.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ON COMMIT</th>
<th>Specifies whether data is transaction- or session-scoped:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>**ON COMMIT {PRESERVE</td>
</tr>
</tbody>
</table>

- DELETE (default) marks the temporary table for transaction-scoped data. Vertica removes all table data after each commit.

- PRESERVE marks the temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica removes all table data when the session ends.

<table>
<thead>
<tr>
<th>load-method</th>
<th>Specifies default load behavior for all DML operations on this table, such as INSERT and COPY, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>AUTO</strong> (default): Initially loads data into WOS, suitable for smaller bulk loads.</td>
</tr>
<tr>
<td></td>
<td><strong>DIRECT</strong>: Loads data directly into ROS containers, suitable for large (&gt;100 MB) bulk loads.</td>
</tr>
</tbody>
</table>
- **TRICKLE**: Loads data only into WOS, suitable for frequent incremental loads.

  For details, see *Choosing a Load Method* in the Administrator's Guide.

**NO PROJECTION**

Prevents Vertica from creating auto-projections for this table. A superprojection is created only when data is explicitly loaded into this table.

**NO PROJECTION** is invalid with the following clauses:

- **ORDER BY**
- **KSAFE**
- Any segmentation clause (*hash-segmentation-clause* or *unsegmented-clause*).

**{INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES**

Specifies default inheritance of schema privileges for this table:

- **EXCLUDE [SCHEMA] PRIVILEGES** (default) disables inheritance of privileges from the schema
- **INCLUDE [SCHEMA] PRIVILEGES** grants the table the same privileges granted to its schema

For more information see *Grant Inherited Privileges*.

**ORDER BY**

**table-column[,...]**

Specifies columns from the SELECT list on which to sort the superprojection that is automatically created for this table. The **ORDER BY** clause cannot include qualifiers ASC or DESC. Vertica always stores projection data in ascending sort order.

If you omit the **ORDER BY** clause, Vertica uses the SELECT list order as the projection sort order.

This option is invalid for external tables.

**segmentation-spec**

Invalid for external tables, specifies how to distribute data for auto-projections of this table. Supply one of the following clauses:

- **hash-segmentation-clause**: Specifies to segment data evenly and distribute across cluster nodes. Vertica recommends segmenting large tables. For details, see *Hash
### Segmentation Clause

- **unsegmented-clause**: Specifies to create an unsegmented projection. For details, see [Unsegmented Clause](#).

If this clause is omitted, Vertica generates auto-projections with default hash segmentation.

### KSAFE \([k-num]\)

Specifies K-safety of auto-projections created for this table, where \(k-num\) must be equal to or greater than system K-safety. If you omit this option, the projection uses the system K-safety level. For general information, see [K-Safety](#) in Vertica Concepts.

**Note**: This option is invalid for external tables.

### column-name-list

Valid only when creating a table from a query (AS `query`), defines column names that map to the query output. If you omit this list, Vertica uses the query output column names. The names in `column-name-list` and queried columns must be the same in number.

For example:

```sql
CREATE TABLE customer_occupations (name, profession)
AS SELECT customer_name, occupation FROM customer_dimension;
```

This clause and the ENCODED BY clause are mutually exclusive. Column name lists are invalid for external tables.

### AS `query`

Creates and loads a table from the results of a query, specified as follows:

```sql
AS [ /*+hint[, hint]*/ ] [ AT epoch ] query
```

You can qualify the AS clause with one or both of the following hints:

- A load method hint: **AUTO, DIRECT, or TRICKLE**

**Note**: The CREATE TABLE statement can also specify a load method. However, this load method applies to load operations only after the table is created.

- **LABEL**
For details, see Creating a Table from a Query in the Administrator's Guide.

<table>
<thead>
<tr>
<th>ENCODED BY</th>
<th>column-ref-list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A list of columns from the source table, where each column is qualified by one or both of the following encoding options:</td>
</tr>
<tr>
<td></td>
<td>- <strong>ACCESSRANK</strong> <em>integer</em>: Overrides the default access rank for a column, useful for prioritizing access to a column. See Prioritizing Column Access Speed in the Administrator's Guide.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ENCODING</strong> <em>encoding-type</em>: Specifies the type of encoding to use on the column. The default encoding type is AUTO.</td>
</tr>
<tr>
<td></td>
<td>This option and <em>column-name-list</em> are mutually exclusive. This option is invalid for external tables</td>
</tr>
</tbody>
</table>

Privileges

The following privileges are required:

- CREATE privileges on the table schema

- If creating a temporary table that includes a named sequence:
  - SELECT privilege on sequence object
  - USAGE privilege on sequence schema

Restrictions

- Queries on temporary tables are subject to the same restrictions on SQL support as persistent tables.

- You cannot add projections to non-empty, global temporary tables (ON COMMIT PRESERVE ROWS). Make sure that projections exist before you load data. See Auto-Projections in the Administrator's Guide.

- While you can add projections for temporary tables that are defined with ON COMMIT DELETE ROWS specified, be aware that you might lose all data.

- Moveout and mergeout operations cannot be used on session-scoped temporary data.
In general, session-scoped temporary table data is not visible using system (virtual) tables.

Temporary tables do not recover. If a node fails, queries that use the temporary table also fail. Restart the session and populate the temporary table.

See Also

- Creating Temporary Tables
- ALTER TABLE
- CREATE TABLE

CREATE TEXT INDEX

Creates a text index used to perform text searches.

Syntax

```
CREATE TEXT INDEX [[database.]schema.]txtindex-name
  ON [schema.]source-table (unique-id, text-field [, column-name, ...])
  [STEMMER {stemmer-name(stemmer-input-data-type)} | NONE]
  [TOKENIZER tokenizer-name(tokenizer-input-data-type)];
```

Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a schema, the table is created in the default schema.</td>
</tr>
</tbody>
</table>
### Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>txtindex-name</code></td>
<td>The text index name.</td>
</tr>
<tr>
<td><code>source-table</code></td>
<td>The source table to index.</td>
</tr>
<tr>
<td><code>unique-id</code></td>
<td>The name of the column in the source table that contains a unique identifier. Any data type is permissible. The column must be the primary key in the source table.</td>
</tr>
<tr>
<td><code>text-field</code></td>
<td>The name of the column in the source table that contains the text field. Valid data types are:</td>
</tr>
<tr>
<td></td>
<td>• CHAR</td>
</tr>
<tr>
<td></td>
<td>• VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• LONG VARCHAR</td>
</tr>
<tr>
<td></td>
<td>• VARBINARY</td>
</tr>
<tr>
<td></td>
<td>• LONG VARBINARY</td>
</tr>
<tr>
<td></td>
<td>Nulls are allowed.</td>
</tr>
<tr>
<td><code>column-name</code></td>
<td>The name of a column or columns to be included as additional columns.</td>
</tr>
<tr>
<td><code>stemmer-name</code></td>
<td>The name of the stemmer</td>
</tr>
<tr>
<td><code>stemmer-input-data-type</code></td>
<td>The input data type of the <code>stemmer-name</code> function.</td>
</tr>
<tr>
<td><code>tokenizer-name</code></td>
<td>Specifies the name of the tokenizer.</td>
</tr>
<tr>
<td><code>tokenizer-input-data-type</code></td>
<td>This value is the input data type of the <code>tokenizer-name</code> function. It can accept any number of arguments.</td>
</tr>
<tr>
<td></td>
<td>If a Vertica Tokenizers is used, then this parameter can be omitted.</td>
</tr>
</tbody>
</table>

### Privileges

The index automatically inherits the query permissions of its parent table. The table owner and dbadmin will be allowed to create and/or modify the indices.
Important: Do not alter the contents or definitions of the text index. If the contents or definitions of the text index are altered, then the results will not appropriately match the source table.

Requirements

- Requires there be a column with a unique identifier set as the primary key.
- The source table must have an associated projection, and must be both sorted and segmented by the primary key.

Behavior

If data within a table is partitioned, then an extra column appears in the text index, showing the partition.

Examples

The following example shows how to create a text index with an additional unindexed column on the table t_log using the CREATE TEXT INDEX statement:

```sql
=> CREATE TEXT INDEX t_log_index ON t_log (id, text, day_of_week);
CREATE INDEX
=> SELECT * FROM t_log_index;
token | doc_id | day_of_week
---|---|---
'catalog' | 1 | Monday
'dbadmin' | 2 | Monday
2014-06-04 | 1 | Monday
2014-06-04 | 2 | Monday
2014-06-04 | 3 | Monday
2014-06-04 | 4 | Monday
2014-06-04 | 5 | Monday
2014-06-04 | 6 | Monday
2014-06-04 | 7 | Monday
2014-06-04 | 8 | Monday
45035996273704966 | 3 | Tuesday
45035996273704968 | 4 | Tuesday
<INFO> | 1 | Tuesday
<INFO> | 6 | Tuesday
<INFO> | 7 | Tuesday
<INFO> | 8 | Tuesday
<WARNING> | 2 | Tuesday
<WARNING> | 3 | Tuesday
<WARNING> | 4 | Tuesday
```
The following example shows a text index, tpart_index, created from a partitioned source table:

```sql
=> SELECT * FROM tpart_index;
```

```
token | doc_id | partition
-------|--------|----------
0      | 4      | 2014
0      | 5      | 2014
11:00:49.568 | 4 | 2014
11:00:49.568 | 5 | 2014
11:00:49.569 | 6 | 2014
<INFO> | 6 | 2014
<WARNING> | 4 | 2014
<WARNING> | 5 | 2014
Database | 6 | 2014
Execute: | 6 | 2014
Object | 4 | 2014
Object | 5 | 2014
[Catalog] | 4 | 2014
[Catalog] | 5 | 2014
'catalog' | 1 | 2013
'dbadmin' | 2 | 2013
0 | 3 | 2013
11:00:49.568 | 1 | 2013
11:00:49.568 | 2 | 2013
11:00:49.568 | 3 | 2013
11:00:49.570 | 7 | 2013
11:00:49.571 | 8 | 2013
45035996273704966 | 3 | 2013
```

See Also

- DROP TEXT INDEX
- DROP TEXT INDEX
CREATE USER

Adds a name to the list of authorized database users.

Note: New users lack default access to schema PUBLIC. Be sure to assign new users USAGE privileges to the PUBLIC schema (GRANT USAGE ON SCHEMA PUBLIC)

Syntax

CREATE USER name [ user-parameter setting[,...] ]

user-parameter

ACCOUNT
GRACEPERIOD
IDENTIFIED BY
IDLESESSIONTIMEOUT
MAXCONNECTIONS
MEMORYCAP
PASSWORD EXPIRE
PROFILE
RESOURCE POOL
RUNTIMECAP
SEARCH_PATH
TEMPSPACECAP

Parameters

| name | Specifies the name of the new user. Names that contain special characters must be double-quoted. To enforce case-sensitivity, use double-quotes. For details on name requirements, see Creating a Database Name and Password. |
| ACCOUNT { LOCK | UNLOCK } | Locks or unlocks a user's access to the database, set to one of the following: |
| | • UNLOCK (default) |
| | • LOCK prevents a new user from logging in. This can be useful when creating an account for a user who does not need immediate access. |
Tip: To automate account locking, set a maximum number of failed login attempts with `CREATE PROFILE`.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GRACEPERIOD limit</strong></td>
<td>Specifies how long a user query can block on any session socket, where <code>limit</code> is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• NONE (default): Removes any grace period previously set on session queries.</td>
</tr>
<tr>
<td></td>
<td>• <code>'interval'</code>: Specifies as an <code>interval</code> the maximum grace period for current session queries, up to 20 days.</td>
</tr>
<tr>
<td></td>
<td>For details, see <a href="#">Handling Session Socket Blocking</a>.</td>
</tr>
<tr>
<td><strong>IDENTIFIED BY 'password'</strong></td>
<td>Sets the new user's password, where <code>password</code> must conform to the password complexity policy set by the user's profile.</td>
</tr>
<tr>
<td></td>
<td>If you supply an empty string or omit this parameter, the user is assigned no password and is not prompted for one when connecting.</td>
</tr>
<tr>
<td></td>
<td>For details, see <a href="#">Password Guidelines</a> and <a href="#">Creating a Database Name and Password</a>.</td>
</tr>
<tr>
<td><strong>IDLESESSIONTIMEOUT limit</strong></td>
<td>The length of time the system waits before disconnecting an idle session, where <code>limit</code> is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• NONE (default): No limit set for this user. If you omit this parameter, no limit is set for this user.</td>
</tr>
<tr>
<td></td>
<td>• <code>'interval'</code>: An interval value, up to one year.</td>
</tr>
<tr>
<td></td>
<td>For details, see <a href="#">Managing Client Connections</a>.</td>
</tr>
<tr>
<td><strong>MAXCONNECTIONS limit</strong></td>
<td>Indicates the maximum number of connections the user can have to the server, where <code>limit</code> is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• NONE (default): No limit. If you omit this parameter, the user can have an unlimited number of connections.</td>
</tr>
<tr>
<td></td>
<td>• <code>'integer'</code> ON NODE: Sets the maximum number of connections to each node to <code>integer</code>.</td>
</tr>
<tr>
<td></td>
<td>• <code>'integer'</code> ON DATABASE: Sets the maximum number of connections across the database cluster to <code>integer</code>.</td>
</tr>
</tbody>
</table>
For details, see Managing Client Connections.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
</table>
| MEMORYCAP $\textit{limit}$ | Specifies how much memory can be allocated to user requests, where $\textit{limit}$ is specified in this format:  
- NONE (default): No limit
- $'\textit{max-expression}'$: A string value that specifies the memory limit, one of the following:
  - $\textit{int\%}$ — Expresses the maximum as a percentage of total memory available to the Resource Manager, where $\textit{int}$ is an integer value between 0 and 100. For example:
    - MEMORYCAP '40%'  
  - $\textit{int\{K|M|G|T\}}$ — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example:
    - MEMORYCAP '10G'

| PASSWORD EXPIRE | Forces immediate expiration of the user's password. The user must change the password on the next login.  
**Note:** PASSWORD EXPIRE has no effect when using external password authentication methods such as LDAP or Kerberos.

| PROFILE $\textit{profile}$ | Assigns a $\textit{profile}$ that controls password requirements for this user, where $\textit{profile}$ is one of the following:
- DEFAULT (default): Assigns the default database profile to this user.
- $\textit{profile-name}$: A profile that is defined by CREATE PROFILE. If you omit this parameter, the user is assigned the default profile.

| RESOURCE POOL $\textit{pool-name}$ | Assigns a default resource pool to this user. The user must also be granted privileges to this pool, unless privileges to the pool are set to PUBLIC.

| RUNTIMECAP $\textit{limit}$ | Specifies how long this user's queries can execute, where
**Limit** is one of the following:

- **NONE** (default): No limit set for this user. If you omit this parameter, no limit is set for this user.
- **'interval'**: An interval value, up to one year.

A query's runtime limit can be set at three levels: the user's runtime limit, the user's resource pool, and the session setting. For more information, see [Setting a Runtime Limit for Queries](https://www.vertica.com/docs/9.0.1/guidesDEVGuidePDF/Vertica-Analytic-Database-GUIDEDEVG_9.0.1.pdf) in the Administrator's Guide.

### SEARCH_PATH path

Specifies the user's default search path that tells Vertica which schemas to search for unqualified references to tables and UDFs, where *path* is one of the following:

- **DEFAULT** (default): Sets the search path as follows:
  
  "$user", public, v_catalog, v_monitor, v_internal

- A comma-delimited list of schemas.


### TEMPSPACECAP limit

Limits how much temporary file storage is available for user requests, where *limit* is one of the following:

- **NONE** (default): No limit
- **'max-expression'**: A string value that specifies the storage limit, one of the following:
  
  - *int%* — Expresses storage as a percentage of total file space is available, where *int* is an integer value between 0 and 100. For example:
    
    TEMPSPACECAP '40%

  - *int{K|M|G|T}* — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example:
    
    TEMPSPACECAP '10G'
Privileges

Superuser

User Name Best Practices

Vertica database user names are logically separate from user names of the operating system in which the server runs. If all the users of a particular server also have accounts on the server's machine, it makes sense to assign database user names that match their operating system user names. However, a server that accepts remote connections might many database users with no local operating system account. In this case, there is no need to connect database and system user names.

Examples

```sql
=> CREATE USER Fred IDENTIFIED BY 'Mxyzptlk';
=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

See Also

- `ALTER USER`
- `DROP USER`

**CREATE VIEW**

Defines a view. Views are read only, so they do not support insert, update, delete, or copy operations.

Syntax

```
CREATE [ OR REPLACE ] VIEW [[database.]schema.]view [ (column[,...]) ]
  [ {INCLUDE|EXCLUDE} [SCHEMA] PRIVILEGES ] AS query
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OR REPLACE</strong></td>
<td>Specifies to overwrite the existing view <code>view-name</code>. If you omit this option and <code>view-name</code> already exists, CREATE VIEW returns an error.</td>
</tr>
<tr>
<td></td>
<td>Any grants assigned to the view before you execute a CREATE OR REPLACE remain on the updated view. See <strong>GRANT (View)</strong>.</td>
</tr>
<tr>
<td><code>[database]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><strong>view</strong></td>
<td>Identifies the view to create, where <code>view-name</code> conforms to conventions described in <strong>Identifiers</strong>. It must also be unique among all names of</td>
</tr>
<tr>
<td></td>
<td>sequences, tables, projections, views, and models within the same schema.</td>
</tr>
<tr>
<td><strong>column[,...]</strong></td>
<td>A list of names to use as view column names. Vertica maps view column names to query columns according to the order of their respective lists. By</td>
</tr>
<tr>
<td></td>
<td>default, the view uses column names as they are specified in the query. Each view can contain up to 1600 columns.</td>
</tr>
<tr>
<td><strong>query</strong></td>
<td>A <strong>SELECT</strong> statement that the temporary view executes. The SELECT statement can reference tables, temporary tables, and other views.</td>
</tr>
<tr>
<td>`{INCLUDE</td>
<td>EXCLUDE}`</td>
</tr>
<tr>
<td><code>[SCHEMA] PRIVILEGES</code></td>
<td>- INCLUDE PRIVILEGES: Inherit schema privileges.</td>
</tr>
<tr>
<td></td>
<td>- EXCLUDE PRIVILEGES: Do not inherit schema privileges.</td>
</tr>
<tr>
<td></td>
<td>For details, see <strong>Grant Inherited Privileges</strong>.</td>
</tr>
</tbody>
</table>
Privileges

See Creating Views

Examples

The following example shows how to create a view that contains data from multiple tables.

```sql
=> CREATE VIEW temp_t0 AS SELECT * from t0_p1 UNION ALL
   SELECT * from t0_p2 UNION ALL
   SELECT * from t0_p3 UNION ALL
   SELECT * from t0_p4 UNION ALL
   SELECT * from t0_p5;
```

See Also

- ALTER VIEW
- CREATE LOCAL TEMPORARY VIEW
- Creating Views
- DROP VIEW
- GRANT (View)
- REVOKE (View)

DELETE

Removes the specified rows from a table and returns a count of the deleted rows. A count of 0 is not an error, but indicates that no rows matched the condition. An unqualified DELETE statement (omits a WHERE clause) removes all rows but leaves intact table columns, projections, and constraints.

DELETE supports subqueries and joins, so you can delete values in a table based on values in other tables.
Syntax

DELETE [ /*+ hint[, hint] */ ] FROM [[database.]schema.]table [ where-clause ]

Parameters

<table>
<thead>
<tr>
<th>/*+ hint [, hint] */</th>
<th>One or both of the following hints:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• DIRECT</td>
</tr>
<tr>
<td></td>
<td>• LABEL</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>table</td>
<td>Any table, including temporary tables.</td>
</tr>
<tr>
<td>where-clause</td>
<td>Specifies which rows to mark for deletion. If you omit this clause, DELETE behavior varies depending on whether the table is persistent or temporary. See below for details.</td>
</tr>
</tbody>
</table>

Privileges

Table owner or user with GRANT OPTION is grantor.

- DELETE privilege on table
- USAGE privilege on the schema of the target table
- SELECT privilege on a table when the DELETE statement includes a WHERE or SET clause that specifies columns from that table.
Restrictions

You cannot execute DELETE on a projection.

Deleting from Persistent Tables

Unlike TRUNCATE TABLE, DELETE does not delete data from disk storage. Rather, it marks rows for deletion in the WOS. These rows are no longer valid in the current epoch. By default, DELETE uses WOS. If WOS fills up, the operation overflows to the ROS.

Deleting from a Temporary Table

DELETE execution on temporary tables varies, depending on whether the table was created with ON COMMIT DELETE ROWS (default) or ON COMMIT PRESERVE ROWS:

- If DELETE contains a WHERE clause that specifies which rows to remove, behavior is identical: DELETE marks the rows for deletion. In both cases, you cannot roll back to an earlier savepoint.

- If DELETE omits a WHERE clause and the table was created with ON COMMIT PRESERVE ROWS, Vertica marks all table rows for deletion. If the table was created with ON COMMIT DELETE ROWS, DELETE behaves like TRUNCATE TABLE and removes all rows from storage.

  Note: If you issue an unqualified DELETE statement on a temporary table created with ON COMMIT DELETE ROWS, Vertica removes all rows from storage but does not end the transaction.

Examples

The following command removes all rows from temporary table temp1:

  => DELETE FROM temp1;

The following command deletes all records from anchor table T where C1 = C2 - C1.

  => DELETE FROM T WHERE C1=C2-C1;
The following command deletes all records from the customer table in the retail schema where the state attribute is in MA or NH:

```sql
=> DELETE FROM retail.customer WHERE state IN ('MA', 'NH');
```

For examples that show how to nest a subquery within a DELETE statement, see Subqueries in UPDATE and DELETE in Analyzing Data.

See Also

- DROP TABLE
- TRUNCATE TABLE
- Removing Table Data
- Best Practices for DELETE and UPDATE

Directed Query Statements

The following statements let you create and manage directed queries:

<table>
<thead>
<tr>
<th>ACTIVATE DIRECTED QUERY</th>
<th>Activates a directed query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE DIRECTED QUERY</td>
<td>Saves an association between an input query and a query that is annotated with optimizer hints.</td>
</tr>
<tr>
<td>DEACTIVATE DIRECTED QUERY</td>
<td>Deactivates one or more directed queries.</td>
</tr>
<tr>
<td>DROP DIRECTED QUERY</td>
<td>Removes a directed query from the database.</td>
</tr>
<tr>
<td>GET DIRECTED QUERY</td>
<td>Returns a list of directed queries stored in the database for a given input query.</td>
</tr>
<tr>
<td>SAVE QUERY</td>
<td>Saves an input query to associate with a custom directed query.</td>
</tr>
</tbody>
</table>
ACTIVATE DIRECTED QUERY

Activates a directed query and makes it available to the query optimizer across all sessions.

Syntax
ACTIVATE DIRECTED QUERY  

Parameters

<table>
<thead>
<tr>
<th>query-name</th>
<th>Identifies the directed query to activate.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To obtain identifiers for directed queries, use GET DIRECTED QUERY, or query the system table V_CATALOG.DIRECTED_QUERIES.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

Activation Life Cycle

After you activate a directed query, it remains active until it is explicitly deactivated by DEACTIVATE DIRECTED QUERY or removed from storage by DROP DIRECTED QUERY. If a directed query is active at the time of database shutdown, Vertica automatically reactivates it when you restart the database.

Examples

See Activating and Deactivating Directed Queries.

CREATE DIRECTED QUERY

Saves an association between an input query and a query that is annotated with optimizer hints.
Syntax

Optimizer-generated

CREATE DIRECTED QUERY OPTIMIZER directedqueryID [COMMENT 'comments']
input-query

User-generated (custom)

CREATE DIRECTED QUERY CUSTOM directedqueryID [COMMENT 'comments']
annotated-query

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIMIZER</td>
<td>Directs the query optimizer to generate an annotated query from input-query, and associate both in the new directed query.</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>Specifies to associate annotated-query with the query previously specified by SAVE QUERY.</td>
</tr>
<tr>
<td>directedqueryID</td>
<td>A unique identifier for the directed query, a string that conforms to conventions described in Identifiers.</td>
</tr>
</tbody>
</table>
| COMMENT 'comments'         | Comments about the directed query, up to 128 characters. Comments can be useful for future reference—for example, explain why a given directed query was created. If you omit this argument, Vertica inserts one of the following comments:  
  - Optimizer-generated directed query  
  - Custom directed query |
| input-query                | The input query to associate with an optimizer-generated directed query. The input query supports only one optimizer hint, IGNORECONST. |
| annotated-query            | A query with embedded optimizer hints to associate with the input query most recently saved with SAVE QUERY. |
Privileges

Superuser

Description

CREATE DIRECTED QUERY associates an input query with a query annotated with optimizer hints. It stores the association under a unique identifier. CREATE DIRECTED QUERY has two variants:

- CREATE DIRECTED QUERY OPTIMIZER directs the query optimizer to generate annotated SQL from the specified input query. The annotated query contains hints that the optimizer can use to recreate its current query plan for that input query.

- CREATE DIRECTED QUERY CUSTOM specifies an annotated query supplied by the user. Vertica associates the annotated query with the input query specified by the last SAVE QUERY statement.

In both cases, Vertica associates the annotated query and input query, and registers their association in the system table V_CATALOG.DIRECTED_QUERIES under query_name.

Caution: Vertica associates a saved query and directed query without checking whether the two are compatible. Be careful to sequence SAVE QUERY and CREATE DIRECTED QUERY CUSTOM so the saved and directed queries are correctly matched.

See Also

Creating Directed Queries

DEACTIVATE DIRECTED QUERY

Deactivates one or more directed queries previously activated by ACTIVATE DIRECTED QUERY.

Syntax

DEACTIVATE DIRECTED QUERY {query-name|input-query}
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query-name</td>
<td>Identifies the directed query to deactivate. To obtain identifiers for directed queries, use GET DIRECTED QUERY, or query the system table V_CATALOG.DIRECTED_QUERIES.</td>
</tr>
<tr>
<td>input-query</td>
<td>The input query of the directed queries to deactivate. Use this argument to deactivate multiple direct queries that map to the same input query.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

DROP DIRECTED QUERY

Removes a directed query from the database. If the directed query is active, Vertica deactivates it before removal.

Syntax

DROP DIRECTED QUERY directedqueryID

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directedqueryID</td>
<td>Identifies the directed query to remove from the database. To obtain identifiers for directed queries, use GET DIRECTED QUERY, or query the system table V_CATALOG.DIRECTED_QUERIES.</td>
</tr>
</tbody>
</table>

Privileges

Superuser
GET DIRECTED QUERY

Returns a list of directed queries that map to the specified input query

Syntax

GET DIRECTED QUERY input-query

Parameters

<table>
<thead>
<tr>
<th>input-query</th>
<th>An input query that is associated with one or more directed queries.</th>
</tr>
</thead>
</table>

Returns

A list of one or more directed queries that map to the specified input query. Each list item includes six fields:

<table>
<thead>
<tr>
<th>query-name</th>
<th>A unique identifier that is associated with this directed query and used by statements such as ACTIVATE DIRECTED QUERY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_active</td>
<td>A Boolean field that specifies whether the directed query is active.</td>
</tr>
<tr>
<td>vertica_version</td>
<td>The Vertica version that was current when this directed query was created.</td>
</tr>
<tr>
<td>comment</td>
<td>A user-supplied comment specified on creation of the direct query.</td>
</tr>
<tr>
<td>creation_date</td>
<td>The creation timestamp of this directed query.</td>
</tr>
<tr>
<td>annotated_query</td>
<td>The annotated query that was saved with CREATE DIRECTED QUERY.</td>
</tr>
</tbody>
</table>

Privileges

None required
Examples

See Getting Directed Queries in the Administrator's Guide.

See Also

V_CATALOG.DIRECTED_QUERIES

SAVE QUERY

Saves an input query to associate with a custom directed query.

Syntax

SAVE QUERY input-query

Parameters

| input-query | The input query to associate with a custom directed query. The input query supports only one optimizer hint, IGNORECONST. |

Privileges

Superuser

Description

SAVE QUERY saves the specified input query for use by the next invocation of CREATE DIRECTED QUERY CUSTOM. CREATE DIRECTED QUERY CUSTOM pairs the saved query with its annotated query argument to create a directed query. Both statements must be issued in the same user session.

The saved query remains available until the one of the following events occurs:
• The next invocation of CREATE DIRECTED QUERY, whether invoked with CUSTOM or OPTIMIZER.

• Another invocation of SAVE QUERY.

• The session ends.

Caution: Vertica associates a saved query with a directed query without checking whether the input and annotated queries are compatible. Be careful to sequence SAVE QUERY and CREATE DIRECTED QUERY CUSTOM so the saved and directed queries are correctly matched.

Examples

See Custom Directed Queries.

**DISCONNECT**

Closes a previously established connection to another Vertica database. You must have previously used the CONNECT statement to perform a COPY FROM VERTICA or EXPORT TO VERTICA statement.

Syntax

```sql
DISCONNECT database-name
```

Parameters

| database-name | The name of the database whose connection to close. |

Privileges

No special permissions required.
Example

```sql
=> DISCONNECT ExampleDB;
DISCONNECT
```

See Also

- CONNECT
- COPY FROM VERTICA
- EXPORT TO VERTICA

**DROP ACCESS POLICY**

Removes an access policy from a column or row. If you attempt to run a DROP TABLE statement on a table that contains an access policy, the following message appears:

```
NOTICE 4927: The AccessPolicy depends on Table tablename
ROLLBACK 3128: DROP failed due to dependencies
DETAIL: Cannot drop Table `tablename` because other objects depend on it
Projection `projection` has column `column_name` as part of its sort order
Projection public.p_1 has column `column_name` as part of its sort order
HINT: Use DROP .. CASCADE to drop or modify the dependent objects
```

Syntax

```
DROP ACCESS POLICY ON `tablename`
FOR { COLUMN `columnname` | ROWS };
```

Parameters

<table>
<thead>
<tr>
<th><code>tablename</code></th>
<th>The name of the table that contains the column access policy you want to remove.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>columnname</code></td>
<td>The name of the column that contains the access policy you want to remove.</td>
</tr>
</tbody>
</table>
Privileges

You must be a dbadmin user to drop an access policy.

Examples

These examples show various cases where you can drop an access policy.

**Drop Column Access Policy from Customer Table for a Specified Column**

=> DROP ACCESS POLICY ON customer FOR COLUMN Customer_Number;

**Drop Access Policy and Table Using the CASCADE Keyword with DROP TABLE**

=> DROP TABLE <tablename> CASCADE;

**Drop Row Access Policy on customer_info table**

=> DROP ACCESS POLICY ON customer_info FOR ROWS;

**DROP AGGREGATE FUNCTION**

Drops a User Defined Aggregate Function (UDAF) from the Vertica catalog.

Syntax

```
DROP AGGREGATE FUNCTION [[database.]schema.]function[, ...]
  ... ( [ [argname] argtype[,,...] ] )
```

Parameters

<table>
<thead>
<tr>
<th><code>[database.]schema</code></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
</tbody>
</table>
If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th><strong>function</strong></th>
<th>Specifies a name of the SQL function to drop. If the function name is schema-qualified, the function is dropped from the specified schema (as noted above).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>argname</strong></td>
<td>Specifies the name of the argument, typically a column name.</td>
</tr>
<tr>
<td><strong>argtype</strong></td>
<td>Specifies the data type for argument(s) that are passed to the function. Argument types must match Vertica type names. See SQL Data Types.</td>
</tr>
</tbody>
</table>

**Privileges**

One of the following:

- Superuser
- Owner

**Notes**

- To drop a function, you must specify the argument types because several functions might share the same name with different parameters.
- Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL functions), Vertica returns an error when those objects are used and not when the function is dropped.

**Example**

The following command drops the ag_avg function:

```sql
=> DROP AGGREGATE FUNCTION ag_avg(numeric);
DROP AGGREGATE FUNCTION
```
See Also

- ALTER FUNCTION (UDF)
- CREATE AGGREGATE FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTION
- Using User-Defined SQL Functions
- Developing User-Defined Extensions (UDxs)
- Aggregate Functions (UDAFs)

**DROP AUTHENTICATION**

Drops an authentication method that has been created but not granted to any users. If you try to drop an authentication method that has been granted to one or more users, the DROP operation fails. To successfully drop an authentication method that is currently granted to a user, use the CASCADE keyword.

**Syntax**

`DROP AUTHENTICATION auth_method_name [ CASCADE ]`

**Parameters**

| `auth_method_name` | Name of the authentication method to drop.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: VARCHAR (not case sensitive)</td>
<td></td>
</tr>
</tbody>
</table>
Privileges

Must have DBADMIN privileges.

Examples

To delete an authentication record for md5_auth, use the following command:

```sql
=> DROP AUTHENTICATION md5_auth;
```

To delete an authentication record for a method that has been granted to a user, use the CASCADE keyword:

```sql
=> CREATE AUTHENTICATION localpwd METHOD 'password' LOCAL;
=> GRANT AUTHENTICATION localpwd TO jsmith;
=> DROP AUTHENTICATION localpwd CASCADE;
```

- ALTER AUTHENTICATION
- CREATE AUTHENTICATION
- GRANT (Authentication)
- REVOKE (Authentication)

**DROP FAULT GROUP**

A drop operation removes the specified fault group and its child fault groups, placing all nodes under the parent of the dropped fault group.

To drop all fault groups, use `ALTER DATABASE..DROP ALL FAULT GROUP` syntax.

To add an orphaned node back to a fault group, you must manually re-assign it to a new or existing fault group using a combination of `CREATE FAULT GROUP` and `ALTER FAULT GROUP..ADD NODE` statements.

Tip: For a list of all fault groups defined in the cluster, query the `V_CATALOG.FAULT_GROUPS` system table.
Syntax

```
DROP FAULT GROUP name
```

Parameters

| name   | Specifies the name of the fault group to drop. |

Privileges

Must be a superuser to drop a fault group.

Example

The following example drops the `group2` fault group from the cluster:

```
exampledb=> DROP FAULT GROUP group2;
```

See Also

- `V_CATALOG.FAULT_GROUPS` and `V_CATALOG.CLUSTER_LAYOUT`
- Fault Groups in the Administrator's Guide
- High Availability With Fault Groups in Vertica Concepts
DROP FUNCTION

Drops a SQL function or User Defined Function (UDF) from the Vertica catalog.

Syntax

```
DROP FUNCTION [[database.]schema.]function[,....] ( [ arg-list ] )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database. schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>function</code></td>
<td>The SQL function to drop.</td>
</tr>
<tr>
<td><code>arg-list</code></td>
<td>A comma-delimited list of arguments as defined for this function when it was created, specified as follows:</td>
</tr>
<tr>
<td></td>
<td><code>[arg-name] arg-type[,....]</code></td>
</tr>
<tr>
<td></td>
<td>where <code>arg-name</code> optionally qualifies <code>arg-type</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>arg-name</code> is typically a column name.</td>
</tr>
<tr>
<td></td>
<td>• <code>arg-type</code> is the name of an SQL data type supported by Vertica.</td>
</tr>
</tbody>
</table>

Privileges

One of the following:

- Superuser
- Schema or function owner
Requirements

- To drop a function, you must specify the argument types because several functions might share the same name with different parameters.

- Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL functions), Vertica returns an error when those objects are used and not when the function is dropped.

Example

The following command drops the `zerowhennull` function in the `macros` schema:

```
=> DROP FUNCTION macros.zerowhennull(x INT);
DROP FUNCTION
```

See Also

- `ALTER FUNCTION (UDF)`
- `CREATE FUNCTION (SQL Functions)`
- `GRANT (User Defined Extension)`
- `REVOKE (User Defined Extension)`
- `USER_FUNCTIONS`
- `Using User-Defined SQL Functions`

DROP SOURCE

Drops a User Defined Load Source function from the Vertica catalog.

Syntax

`DROP SOURCE [[database.]schema.]source()`
Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>

| source() | Specifies the source function to drop. You must append empty parentheses to the function name. |

Privileges

One of the following:

- Superuser
- Source function owner

Example

The following command drops the curl source function:

```sql
=> DROP SOURCE curl();
DROP SOURCE
```

See Also

- ALTER FUNCTION (UDF)
- CREATE SOURCE
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
• USER_FUNCTIONS
• Load (UDLs)

DROP FILTER

Drops a User Defined Load Filter function from the Vertica catalog.

Syntax

DROP FILTER [[database.]schema.]filter()

Parameters

| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:
| myschema.thisDbObject |
| filter()          | Specifies the filter function to drop. You must append empty parentheses to the function name. |

Privileges

Only the superuser or owner can drop the filter function.

Example

The following command drops the Iconverter filter function:

=> drop filter Iconverter();
DROP FILTER
See Also

- ALTER FUNCTION (UDF)
- CREATE FILTER
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Load (UDLs)

DROP PARSER

Drops a User Defined Load Parser function from the Vertica catalog.

Syntax

DROP PARSER([database.]schema.]parser())

Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>parser()</td>
<td>The name of the parser function to drop. You must append empty parentheses to the function name.</td>
</tr>
</tbody>
</table>

Privileges

Only the superuser or owner can drop the parser function.
Example

The following command drops the BasicIntegerParser parser function:

```
=> DROP PARSER BasicIntegerParser();
```

See Also

- ALTER FUNCTION (UDF)
- CREATE PARSER
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- USER_FUNCTIONS
- Load (UDLs)

**DROP LIBRARY**

Removes a shared library from the database. The library file is deleted from managed directories on the Vertica nodes. The user defined functions (UDFs) in the library are no longer available. See Developing User-Defined Extensions (UDxs) in Extending Vertica for details.

**Syntax**

```
DROP LIBRARY [[database.]schema.]library [CASCADE]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
</tbody>
</table>
If you specify a database, it must be the current database.

<table>
<thead>
<tr>
<th>Library</th>
<th>The name of the library to drop, the same name used in CREATE LIBRARY to load the library.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>Drops any functions that were defined using the library. DROP LIBRARY fails if CASCADE is omitted and one or more UDFs use the target library.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Example**

To drop the library MyFunctions:

```sql
=> DROP LIBRARY MyFunctions CASCADE;
```

**DROP MODEL**

Removes a model from the Vertica database. You can drop multiple models at one time.

**Syntax**

```
DROP MODEL [IF EXISTS] [[db-name.][schema.]model-name
```

**Parameters**

| IF EXISTS | Specifies not to report an error if one or more of the models to drop does not exist. This clause is useful in SQL scripts where you want to drop a model if it exists before recreating it. |
Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:

```sql
myschema
```

The model to drop.

### Privileges

Any user who creates a model can drop or alter his or her own model. If you are the dbadmin user, you can drop or alter any model in the database.

### Examples

This example shows how you can remove a model.

```sql
=> DROP MODEL mySVMModel;
DROP MODEL
```

### DROP NETWORK INTERFACE

Removes a network interface from Vertica. You can use the CASCADE option to also remove the network interface from any node definition. (See Identify the Database or Nodes Used for Import/Export for more information.)

#### Syntax

```sql
DROP NETWORK INTERFACE network-interface-name [CASCADE]
```

#### Parameters

The parameters are defined as follows:

| network-interface-name | The network interface you want to remove. |
Privileges

Must be a superuser to drop a network interface.

Examples

This example shows how to drop a network interface.

```sql
=> DROP NETWORK INTERFACE myNetwork;
```

**DROP NOTIFIER**

Drops a push-based notifier created by `CREATE NOTIFIER`.

Syntax

```
DROP NOTIFIER notifier-name
```

Parameters

<table>
<thead>
<tr>
<th>notifier-name</th>
<th>This notifier's unique identifier.</th>
</tr>
</thead>
</table>

**DROP PROCEDURE**

Removes an external procedure from Vertica. Only the reference to the procedure is removed. The external file remains in the `database/procedures` directory of each database node.

Syntax

```
DROP PROCEDURE [[database.]schema.]procedure( [ arg-list ] )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>procedure</td>
<td>Specifies the procedure to drop.</td>
</tr>
<tr>
<td>arg-List</td>
<td>A comma-delimited list of arguments defined for this procedure when it was created, specified as follows:</td>
</tr>
<tr>
<td></td>
<td><code>[arg-name] arg-type[, ...]</code></td>
</tr>
<tr>
<td></td>
<td>where <code>arg-name</code> optionally qualifies <code>arg-type</code>. If no arguments are defined for this procedure, specify empty parentheses.</td>
</tr>
</tbody>
</table>

Privileges

- Superuser or procedure owner
- USAGE privilege on schema, or schema owner

Example

```
=> DROP PROCEDURE helloplanet(arg1 varchar);
```

See Also

CREATE PROCEDURE
**DROP PROFILE**

Removes a profile from the database. Only the superuser can drop a profile.

**Syntax**

```sql
DROP PROFILE name [, ...] [ CASCADE ]
```

**Parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>The name of one or more profiles (separated by commas) to be removed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>Moves all users assigned to the profile or profiles being dropped to the DEFAULT profile. If you do not include CASCADE in the DROP PROFILE command and a targeted profile has users assigned to it, the command returns an error.</td>
</tr>
</tbody>
</table>

**Privileges**

Must be a superuser to drop a profile.

**Notes**

You cannot drop the DEFAULT profile.

**Example**

```sql
=> DROP PROFILE sample_profile;
```
See Also

- ALTER PROFILE
- CREATE PROFILE

**DROP PROJECTION**

Marks a projection to drop from the catalog so it is unavailable to user queries.

**Syntax**

```
DROP PROJECTION { [[database.]schema.]projection [ , ... ] } ... [ RESTRICT | CASCADE ]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `[database.]schema` | Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:  
  `myschema.thisDbObject`  
  If you specify a database, it must be the current database. |
| `projection`    | Specifies a projection to drop:  
  - If the projection is unsegmented, all projection replicas in the database cluster are dropped.  
  - If the projection is segmented, drop all buddy projections by specifying the projection base name. You can also specify the name of a specific buddy projection as long as dropping it so does not violate system K-safety.  
    See Projection Naming for projection name conventions. |
<p>| <code>RESTRICT | CASCADE</code> | Specifies whether to drop the projection when it contains |</p>
<table>
<thead>
<tr>
<th>objects:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• RESTRICT (default): Drop the projection only if it contains no objects.</td>
</tr>
<tr>
<td>• CASCADE: Drop the projection even if it contains objects.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser, or owner of the anchor table

**Restrictions**

The following restrictions apply to dropping a projection:

- The projection cannot be the anchor table's superprojection.
- You cannot drop a buddy projection if doing so violates system K-safety.
- Another projection must be available to enforce the same primary or unique key constraint.

**See Also**

- CREATE PROJECTION
- DROP TABLE
- GET_PROJECTIONS
- GET_PROJECTION_STATUS
- MARK DESIGN KSAFE
- Adding Nodes
**DROP RESOURCE POOL**

Drops a user-created resource pool. All memory allocated to the pool is returned back to the GENERAL pool.

**Syntax**

```
DROP RESOURCE POOL pool-name
```

**Parameters**

| pool-name | Specifies the name of the resource pool to be dropped. |

**Privileges**

Must be a superuser to drop a resource pool.

**Dropping a Secondary Pool**

If you try to drop a resource pool that is a secondary pool for another resource pool, Vertica returns an error. The error lists the resource pools that depend on the secondary pool you tried to drop. To drop a secondary resource pool, first set the CASCADE TO parameter to DEFAULT on the primary resource pool, and then drop the secondary pool.

For example, you can drop resource pool rp2, which is a secondary pool for rp1, as follows:

```
=> ALTER RESOURCE POOL rp1 CASCADE TO DEFAULT;
=> DROP RESOURCE POOL rp2;
```

**Transferring Resource Requests**

Any requests queued against the pool are transferred to the GENERAL pool according to the priority of the pool compared to the GENERAL pool. If the pool’s priority is higher than the
GENERAL pool, the requests are placed at the head of the queue; otherwise the requests are placed at the end of the queue.

Any users who are using the pool are switched to use the GENERAL pool with a NOTICE:

| NOTICE: Switched the following users to the General pool: username |

DROP RESOURCE POOL returns an error if the user does not have permission to use the GENERAL pool. Existing sessions are transferred to the GENERAL pool regardless of whether the session's user has permission to use the GENERAL pool. This can result in additional user privileges if the pool being dropped is more restrictive than the GENERAL pool. To prevent giving users additional privileges, follow this procedure to drop restrictive pools:

1. Revoke the permissions on the pool for all users.
2. Close any sessions that had permissions on the pool.
3. Drop the resource pool.

Examples

The following command drops the resource pool that was created for the CEO:

```sql
=> DROP RESOURCE POOL ceo_pool;
```

See Also

- ALTER RESOURCE POOL
- CREATE RESOURCE POOL
- Managing Workloads

**DROP ROLE**

Removes a role from the database. Only the database superuser can drop a role.

Use the CASCADE option to drop a role that is assigned to one or more users or roles.
NOTE: You cannot use the DROP ROLE command on a role added to the Vertica database with the LDAPLink service.

Syntax

```
DROP ROLE role [CASCADE];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>The name of the role to drop</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revoke the role from users and other roles before dropping the role</td>
</tr>
</tbody>
</table>

Privileges

Must be a superuser to drop a role.

Examples

```
=> DROP ROLE appadmin;
NOTICE: User bob depends on Role appadmin
ROLLBACK: DROP ROLE failed due to dependencies
DETAIL: Cannot drop Role appadmin because other objects depend on it
HINT: Use DROP ROLE ... CASCADE to remove granted roles from the dependent users/roles
=> DROP ROLE appadmin CASCADE;
DROP ROLE
```

See Also

- ALTER ROLE RENAME
- CREATE ROLE
DROP SCHEMA

Permanently removes a schema from the database. Be sure that you want to remove the schema before you drop it, because DROP SCHEMA is an irreversible process. Use the CASCADE parameter to drop a schema containing one or more objects.

Syntax

DROP SCHEMA [database.]schema[,...] [ CASCADE | RESTRICT ]

Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Name of the schema to drop. If you specify a database, it must be the current database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>Specifies to drop the schema and all objects in it, regardless of who owns those objects.</td>
</tr>
<tr>
<td></td>
<td>Caution: Objects in other schemas that depend on objects in the dropped schema—for example, user-defined functions—also are silently dropped.</td>
</tr>
<tr>
<td>RESTRICT</td>
<td>Drops the schema only if it is empty (default).</td>
</tr>
</tbody>
</table>

Privileges

Schema owner
## Restrictions

- You cannot drop the PUBLIC schema.
- If a user is accessing an object within a schema that is in the process of being dropped, the schema is not deleted until the transaction completes.
- Canceling a DROP SCHEMA statement can cause unpredictable results.

## Examples

The following example drops schema S1 only if it doesn't contain any objects:

```sql
=> DROP SCHEMA S1;
```

The following example drops schema S1 whether or not it contains objects:

```sql
=> DROP SCHEMA S1 CASCADE;
```

## DROP SEQUENCE

Removes the specified named sequence number generator.

### Syntax

```sql
DROP SEQUENCE [[database.]schema.]sequence[, ...]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
</tbody>
</table>
**sequence**  |  Specifies the sequence to drop.

**Privileges**

One of the following:

- Superuser
- Sequence or schema owner

**Restrictions**

- For sequences specified in a table's default expression, the default expression fails the next time you try to load data. Vertica does not check for these instances.
- DROP SEQUENCE does not support the CASCADE keyword. Sequences used in a default expression of a column cannot be dropped until all references to the sequence are removed from the default expression.

**Example**

The following command drops the sequence named *sequential*.

```
=> DROP SEQUENCE sequential;
```

**See Also**

- ALTER SEQUENCE
- CREATE SEQUENCE
- CURRVAL
- GRANT (Sequence)
- NEXTVAL
Sequence Types

Named Sequence Privileges

**DROP SUBNET**

Removes a subnet from Vertica. You can use the CASCADE option to also remove the subnet from any database definition. (See Identify the Database or Nodes Used for Import/Export for more information.)

**Syntax**

```
DROP SUBNET subnet-name [CASCADE]
```

**Parameters**

The parameters are defined as follows:

<table>
<thead>
<tr>
<th>subnet-name</th>
<th>The subnet you want to remove.</th>
</tr>
</thead>
</table>

If you remove a subnet, be sure your database is not configured to allow export on the public subnet. (See Identify the Database or Nodes Used for Import/Export for more information.)

**Privileges**

Must be a superuser to drop a subnet.

**Examples**

This example shows how to remove a subnet.

```
=> DROP SUBNET mySubnet;
```
**DROP TABLE**

Removes a table and its projections. When you run DROP TABLE, the change is auto-committed.

**Syntax**

```
DROP TABLE [ IF EXISTS ] [[database.]schema.]table [ , ... ] [ CASCADE ]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF EXISTS</td>
<td>Specifies not to report an error if one or more of the tables to drop does not exist. This clause is useful in SQL scripts where you want to drop a table if it exists before recreating it.</td>
</tr>
<tr>
<td>[ schema ]</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>table</td>
<td>The table to drop.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Specifies to drop all projections for this table before the table is dropped. CASCADE is optional if only auto-projections are associated with this table; otherwise it is required. This option is not valid for external tables.</td>
</tr>
</tbody>
</table>

**Privileges**

- Table owner with USAGE privilege on table's schema
- Schema owner
Requirements

- Do not cancel an executing DROP TABLE. Doing so can leave the database in an inconsistent state.
- Check that the target table is not in use, either directly or indirectly—for example, in a view.
- If you drop and restore a table that is referenced by a view, the new table must have the same name and column definitions.

Examples

See Dropping Tables in the Administrator's Guide.

See Also

- DROP PROJECTION
- TRUNCATE TABLE

**DROP TEXT INDEX**

Drops a text index used to perform text searches.

Syntax

```
DROP TEXT INDEX [ IF EXISTS ] [[database.]schema.]idx-table
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF EXISTS</td>
<td>If specified, DROP TEXT INDEX does not report an error if one or more of the tables to be dropped does not exist. You can use this clause in SQL scripts where you want to drop a table, if it exists,</td>
</tr>
</tbody>
</table>
| **[database.]schema** | Specifies a schema, by default public. If **schema** is any schema other than public, you must supply the schema name. For example:

```
myschema.thisDbObject
```

If you specify a database, it must be the current database. |
| **idx-table** | Specifies the text index name. When using more than one schema, specify the schema that contains the index in the DROP TEXT INDEX statement. |

### Privileges

Table owner and dbadmin with USAGE privilege on schema that contains the table or schema owner.

### Behavior

When a source table is dropped that has a text index associated with it, the text index is also dropped.

### Examples

The following example drops the text index `t_text_index`:

```
=> DROP TEXT INDEX t_text_index;
DROP INDEX
```

### See Also

- [Using Text Search](#)
- [CREATE TEXT INDEX](#)
DROP TRANSFORM FUNCTION

Drops a user-defined transform function (UDTF) from the Vertica catalog.

Syntax

```
DROP TRANSFORM FUNCTION [[database.]schema.]function( [ arg-list ] )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>function</td>
<td>Specifies the transform function to drop.</td>
</tr>
<tr>
<td>arg-list</td>
<td>A comma-delimited list of arguments as defined for this function when it was created, specified as follows:</td>
</tr>
<tr>
<td></td>
<td>[arg-name] arg-type[,...]</td>
</tr>
<tr>
<td></td>
<td>If multiple functions share the same name, specify the argument types of the function you wish to drop.</td>
</tr>
<tr>
<td></td>
<td>• arg-name is typically a column name. You (optionally) use an arg-name to qualify an arg-type.</td>
</tr>
<tr>
<td></td>
<td>• arg-type is the name of an SQL data type supported by Vertica.</td>
</tr>
<tr>
<td></td>
<td>You do not need to include an arg-list when dropping a polymorphic function.</td>
</tr>
</tbody>
</table>

Privileges

One of the following:
- Superuser
- Schema or function owner

Example

The following command drops the `tokenize` UDF in the `macros` schema:

```sql
=> DROP TRANSFORM FUNCTION macros.tokenize(varchar);
```

The following command drops the `Pagerank` polymorphic function in the `online` schema:

```sql
=> DROP TRANSFORM FUNCTION online.Pagerank();
```

See Also

CREATE TRANSFORM FUNCTION

**DROP USER**

Removes a name from the list of authorized database users.

NOTE: You cannot use the DROP USER command on a user added to the Vertica database with the LDAPLink service.

Syntax

```sql
DROP USER name [ , ... ] [ CASCADE ]
```

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Specifies the name or names of the user to drop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>[Optional] Drops all user-defined objects created by the user dropped, including</td>
</tr>
</tbody>
</table>
Privileges

Must be a superuser to drop a user.

Examples

DROP USER <username> fails if objects exist that were created by the user, such as schemas, and tables and their associated projections:

```sql
=> DROP USER user1;
NOTICE: Table T_tbd1 depends on User user1
ROLLBACK: DROP failed due to dependencies
DETAIL: Cannot drop User user1 because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too
```

DROP USER <name> CASCADE succeeds regardless of any pre-existing user-defined objects. The statement forcibly drops all user-defined objects, such as schemas, tables and their associated projections:

```sql
=> DROP USER user1 CASCADE;
```

Caution: Tables owned by the user being dropped cannot be recovered after you issue DROP USER CASCADE.

DROP USER <username> succeeds if no user-defined objects exist (no schemas, tables or projections defined by the user):

```sql
=> CREATE USER user2;
CREATE USER
=> DROP USER user2;
DROP USER
```

See Also

- ALTER USER
- CREATE USER
DROP VIEW

Removes the specified view. Vertica does not check for dependencies on the dropped view. After dropping a view, other views that reference it fail.

If you drop a view and replace it with another view or table with the same name and column names, other views that reference that name use the new view. If you change the column data type in the new view, the server coerces the old data type to the new one if possible; otherwise, it returns an error.

Syntax

DROP VIEW [\[[database.]schema.]view [, ... ]

Parameters

<table>
<thead>
<tr>
<th>[database.]schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td>view</td>
<td>Specifies the view to drop.</td>
</tr>
</tbody>
</table>

Privileges

One of the following

- View owner and USAGE privileges
- Schema owner
Examples

```sql
=> DROP VIEW myview;
```

**END**

Ends the current transaction and makes all changes that occurred during the transaction permanent and visible to other users.

**Syntax**

```
COMMIT [ WORK | TRANSACTION ]
```

**Parameters**

<table>
<thead>
<tr>
<th>WORK</th>
<th>TRANSACTION</th>
<th>Have no effect; they are optional keywords for readability.</th>
</tr>
</thead>
</table>

**Privileges**

No special permissions required.

**Notes**

`COMMIT` is a synonym for `END`.

**Examples**

This example shows how to end a transaction.

```sql
=> CREATE TABLE sample_table (a INT);
=> INSERT INTO sample_table (a) VALUES (1);
```
See Also

- Transactions
- Creating Transactions
- BEGIN
- ROLLBACK
- START TRANSACTION

**EXPLAIN**

Returns a formatted description of the Vertica optimizer's plan for executing the specified statement.

**Syntax**

```
EXPLAIN [/*+ ALLNODES */] [explain-options] sql-statement
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>+ALLNODES</em>/</td>
<td>Specifies to create a query plan that assumes all nodes are active, not valid with LOCAL option.</td>
</tr>
<tr>
<td>explain-options</td>
<td>One or more EXPLAIN options, specified in the order shown:</td>
</tr>
<tr>
<td></td>
<td>[ LOCAL ] [ VERBOSE ] [ JSON ] [ ANNOTATED ]</td>
</tr>
<tr>
<td></td>
<td>• LOCAL: On a multi-node database, shows the local query plans assigned to each node, which together comprise the total (global) query plan. If you omit this option, Vertica shows only</td>
</tr>
</tbody>
</table>
the global query plan. Local query plans are shown only in DOT language source, which can be rendered in Graphviz.

This option is incompatible with the hint /*+ALL NODES*/. If you specify both, EXPLAIN returns with an error.

- **VERBOSE**: Increases the level of detail in the rendered query plan.
- **JSON**: Renders the query plan in JSON format. This option is compatible only with VERBOSE.
- **ANNOTATED**: Embeds optimizer hints that encapsulate the query plan for this query. Vertica uses these hints to create directed queries. For more information, see Directed Queries in the Administrator's Guide. This option is compatible with LOCAL and VERBOSE.

### sql-statement

A query or DML statement—for example, SELECT, INSERT, UPDATE, COPY, and MERGE.

### Privileges

The same privileges required by the specified statement.

### Requirements

The following requirements apply to EXPLAIN's ability to produce useful information:


- EXPLAIN produces useful output only if projections are available for the queried tables.

- Qualifier options must be specified in the order shown earlier, otherwise EXPLAIN returns with an error. If an option is incompatible with any preceding options, EXPLAIN ignores them.
See Also

EXPLAIN Output Options in the Administrator's Guide

**EXPORT TO PARQUET**

Exports a table, columns from a table, or query results to files in the Parquet format. You can use an OVER() clause to partition the data before export. Partitioning data can improve query performance by enabling partition pruning; see Improving Query Performance for Data Stored in HDFS.

You can export data stored in Vertica in ROS format and data from external tables.

EXPORT TO PARQUET returns the number of rows written.

During the export, Vertica writes files to a temporary directory in the same location as the destination and renames the directory when the export is complete. Do not attempt to use the files in the temporary directory.

**Syntax**

```
EXPORT TO PARQUET ( directory = 'directory'
    [, compression = 'compression-type' ]
    [, rowGroupSizeMB = size ]
    [ OVER (over-clause ) ]
AS SELECT query-expression;
```

**Parameters**

| **directory** | The destination directory for the Parquet files. The directory must not exist, and the current user must have permission to write it. The directory can be in HDFS or on the local file system. |
| **compression-type** | Column compression type, one of Snappy or Uncompressed. The default is Snappy. |
| **size** | The uncompressed size of exported row groups, in MB. The maximum value is 512 and the default is 64. The row groups in the |
exported files are smaller because Parquet files are compressed on write. For best performance, set size to be smaller than the HDFS block size.

Arguments

| over-clause | Specifies how to partition table data using PARTITION BY. Within partitions you can sort by using ORDER BY. See SQL Analytics. This clause may contain column references but not expressions. This clause, if present, can contain only column references, not an expression. If you partition data, Vertica creates a Hive-style partition directory structure. See Using Partition Columns for a description of the structure. If you omit this clause, Vertica optimizes for maximum parallelism. |
| query-expression | Specifies the data to be exported. See SELECT for the syntax. You must provide an alias column label for selected column targets that are expressions. |

Privileges

- SELECT privileges on the source table
- USAGE privileges on source table schema
- Write privileges for the destination directory

Data Types

EXPORT TO PARQUET does not support the following data types:

- INTERVAL
- TIME/TIMETZ
- **UUID Data Type**

  Decimal precision must be \( \leq 38 \).

  Vertica does not convert \text{TIMESTAMP} values to UTC. To avoid problems arising from time zones, use \text{TIMESTAMPZ} instead of \text{TIMESTAMP}.

  The exported Hive types might not be identical to the Vertica types. For example, a Vertica INT is exported as a Hive BIGINT. When defining Hive external tables to read exported data, you might have to adjust column definitions.

  This operation exports raw Flex columns as binary data.

**Output Files**

If you specify a directory in the local file system that is not shared storage, Vertica distributes the files among nodes according to how the export is partitioned. To write all local files on one node, use an empty \text{OVER}() clause.

If you specify a directory in the local file system, you must have a USER storage location.

Output file names follow the pattern: [8-character hash]-[node name]-[thread_id].parquet.

Vertica does not support simultaneous exports to the same directory in HDFS. The results are undefined.

Parquet files exported to a local filesystem by any Vertica user are owned by the Vertica superuser. Parquet files exported to HDFS are owned by the Vertica user who exported the data.

**Examples**

The following example demonstrates exporting all columns from the T1 table in the public schema, using Snappy compression (the default).

```sql
=> EXPORT TO PARQUET(directory = 'hdfs:///user1/data')
    AS SELECT * FROM public.T1;

Rows Exported
-----------
  87436
(1 row)
```

The following example demonstrates exporting the results of a query using more than one table.
The following example demonstrates partitioning and exporting data. EXPORT TO PARQUET first partitions the data on 'b' and then, within each partition, sorts by 'd'.

```
=> EXPORT TO PARQUET(directory='hdfs:///user2/data')
   OVER(PARTITION BY b ORDER BY d) AS SELECT b, d FROM public.T2;
```

Rows Exported
-----------------
120931
(1 row)

The following example uses an alias column label for a selected column target that is an expression.

```
=> EXPORT TO PARQUET(directory='hdfs:///user3/data')
   OVER(ORDER BY col1) AS SELECT col1 + col1 AS A, col2
   FROM public.T3;
```

Rows Exported
-----------------
14336
(1 row)

**EXPORT TO VERTICA**

Exports an entire table, columns from a table, or query results to another Vertica database. Exported data is always written in AUTO mode.

**Important:** The source database can be one major release behind the target database.

**Syntax**

```
EXPORT TO VERTICA database.[schema.]target-table
   ... [ ( target-column[,...] ) ]
   ... { AS SELECT query-expression | FROM [schema.]source-table[ ( source-column[,...] ) ] };
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>A string containing the name of the database to receive the exported data. There must be an active connection to this database for the export to succeed.</td>
</tr>
<tr>
<td>[scope. ]<strong>target-table</strong></td>
<td>The table to store the exported data (schema specification is optional). This table must already exist.</td>
</tr>
<tr>
<td><strong>target-column</strong></td>
<td>A list of columns in the target table to store the exported data.</td>
</tr>
<tr>
<td><strong>query-expression</strong></td>
<td>Specifies the data to be exported. See SELECT for the syntax.</td>
</tr>
<tr>
<td>[scope. ]<strong>source-table</strong></td>
<td>The table that contains the data to export.</td>
</tr>
<tr>
<td><strong>source-column</strong></td>
<td>A list of the columns in the source table to export. If present, only these columns are exported.</td>
</tr>
</tbody>
</table>

Privileges

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

Connecting to the Target Database

Before you can export data to another database, you must establish a connection to the target database with CONNECT. See Exporting Data to Another Vertica Database for details.

By default, EXPORT TO VERTICA exports data to another database over the Vertica private network. Connecting to a public network requires some configuration. For details about exporting data across a public network, see Using Public and Private IP Networks.
The export operation fails if either side of the connection is a single-node cluster installed to localhost, or you do not specify a host name or IP address.

Source and Target Column Mapping

You can optionally name a subset of source and target columns to participate in the export operation. EXPORT TO VERTICA tries to match columns in the source table with corresponding columns in the destination table. If you do not supply a list of source and destination columns, EXPORT TO VERTICA tries to match columns in the source table with corresponding columns in the destination table. Auto-projections for the target table are similar to the projections for the source table.

The following table compares the different combinations of naming source and target columns, and the requirements that pertain to each option.

<table>
<thead>
<tr>
<th>Omit source columns</th>
<th>Specify source columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omit target columns</td>
<td>Match all columns in source table to columns in target table. The number of columns in the two tables can differ, but the target table must have at least as many columns as the source table.</td>
</tr>
<tr>
<td>Specify target columns</td>
<td>Match source columns to the named target columns. The number of named target columns must equal the number of columns in the source table.</td>
</tr>
</tbody>
</table>

Node Failure During EXPORT

See Handling Node Failure During Copy/Export in the Administrator's Guide.

Examples

See Exporting Data to Another Vertica Database in the Administrator's Guide.
See Also

COPY FROM VERTICA
**GRANT Statements**

GRANT statements grant privileges on database objects to specific users.

**GRANT (Authentication)**

Associates (or *grants*) an authentication record to one or more users or roles.

**Syntax**

```
GRANT AUTHENTICATION auth-method-name TO
   { Public | user_or_role | user_or_role1, user_or_role2, user_or_role3, ... }
```

**Parameters**

<table>
<thead>
<tr>
<th>auth_method_name</th>
<th>Name of the authentication method you want to associate with one or more users or roles. Type: VARCHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_or_role, user_or_role1, user_or_role2, user_or_role3, ...</td>
<td>Names of users or roles with whom you want to associate the authentication method. Type: VARCHAR.</td>
</tr>
</tbody>
</table>

**Privileges**

Must have DBADMIN privileges.

**Examples**

This example uses a GRANT AUTHENTICATION statement to associate v_ldap authentication with user jsmith:

```sql
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```
This example uses a GRANT AUTHENTICATION statement to associate v_gss authentication to the role DBprogrammer:

```sql
=> CREATE ROLE DBprogrammer;
=> GRANT AUTHENTICATION v_gss to DBprogrammer;
```

This example sets the default client authentication method to v_localpwd:

```sql
=> GRANT AUTHENTICATION v_localpwd TO Public;
```

See Also

- ALTER AUTHENTICATION
- CREATE AUTHENTICATION
- DROP AUTHENTICATION
- REVOKE (Authentication)

GRANT (Database)

Grants the right to create schemas within the database to a user or role. By default, only the superuser has the right to create a database schema.

Syntax

```
GRANT { ...
   | { CREATE [, ...]
   | { TEMP } }
   | { ALL [ PRIVILEGES ]
   | { CONNECT } }
   | ON DATABASE database-name [ , ...]
   | TO { username | rolename } [ , ...]
   | [ WITH GRANT OPTION ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Allows the user to create schemas within the specified database.</td>
</tr>
<tr>
<td>TEMP</td>
<td>Allows the user to create temporary tables in the database.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Allows the user to connect to a database.</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>ALL</td>
<td>Applies to all privileges.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>database-name</td>
<td>Identifies the database in which to grant the privilege.</td>
</tr>
<tr>
<td>username</td>
<td>Grants the privilege to the specified user or role.</td>
</tr>
<tr>
<td>rolename</td>
<td></td>
</tr>
<tr>
<td>WITH GRANT OPTION</td>
<td>Allows the recipient of the privilege to grant it to other users.</td>
</tr>
</tbody>
</table>

**Example**

The following example grants user Fred the right to create schemas on vmartdb.

```sql
=> GRANT CREATE ON DATABASE vmartdb TO Fred;
```

**See Also**

- [REVOKE (Database)](Vertica Analytic Database (9.0.x) Page 3551 of 6180)
- [Granting and Revoking Privileges](Vertica Analytic Database (9.0.x) Page 3551 of 6180)

**GRANT (Library)**

Grants privileges on a library to a database user or role. Only the superuser can grant privileges to a library. To execute functions inside the library, a user must have separate EXECUTE privileges for those functions.

For example, when working with the Connector Framework Service you may need to grant a user usage privileges to a library to be able to set UDSession parameters. For more information see [Implementing CFS](Vertica Analytic Database (9.0.x) Page 3551 of 6180).

**Syntax**

```
grant { usage | all } on library [[database.]schema.]library [, ...]
  to { username | role | public } [, ...]
```
## Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{ USAGE</td>
<td>ALL }`</td>
</tr>
<tr>
<td></td>
<td>- USAGE</td>
</tr>
<tr>
<td></td>
<td>- ALL</td>
</tr>
<tr>
<td><code>[ database. ] schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>library</code></td>
<td>The library on which to grant the privilege. If using more than one schema, you must specify the schema that where the library resides.</td>
</tr>
<tr>
<td>`{ username</td>
<td>role</td>
</tr>
<tr>
<td></td>
<td>- <code>username</code>—Assigns privileges to a specific user</td>
</tr>
<tr>
<td></td>
<td>- <code>role</code>—Assigns privileges to a role</td>
</tr>
<tr>
<td></td>
<td>- PUBLIC—Assigns privileges to all users and roles.</td>
</tr>
</tbody>
</table>

### Example

The following command grants USAGE privileges on the `idolLib` library in the `v_idol` schema to `user1`:

```sql
=> GRANT USAGE ON LIBRARY v_idol.Idollib TO user1;
```

### See Also

- `REVOKE (Library)`
- `Granting and Revoking Privileges`
GRANT (Model)

Grants access to models to a database user or role.

Syntax

```
GRANT {ALL | USAGE }
... ON MODEL [ [ db-name.]schema.]modelname [ , ... ]
... TO { username | role | PUBLIC}
... [ WITH GRANT OPTION ]
```

Parameters

<table>
<thead>
<tr>
<th>ALL</th>
<th>USAGE</th>
<th>Applies to all privileges.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[db-name.]schema.database-name</td>
<td>Specifies a schema. If multiple schemas are defined in the database, include the schema name. For example:</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>role</td>
<td>PUBLIC</td>
</tr>
<tr>
<td>WITH GRANT OPTION</td>
<td>Allows the user to grant the same privileges to other users.</td>
<td></td>
</tr>
</tbody>
</table>

Example

This example grants USAGE privileges on the mySvmClassModel model to user1:

```
=> GRANT USAGE ON MODEL mySvmClassModel TO user1;
```

See Also

- REVOKE (Model)
- Managing Model Security
GRANT (Procedure)

Grants privileges on a procedure to a database user or role. Only the superuser can grant privileges to a procedure. To grant privileges to a schema containing the procedure, users must have USAGE privileges. See GRANT (Schema).

External procedures that you create with CREATE PROCEDURE are always run with Linux dbadmin privileges. If a dbadmin or pseudosuperuser grants a non-dbadmin permission to run a procedure using GRANT (Procedure), be aware that the non-dbadmin user runs the procedure with full Linux dbadmin privileges.

Syntax

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] } 
ON PROCEDURE [[database.]schema.]procedure [,...] 
( [ argname ] argtype [,,... ] ) 
TO { username | role | PUBLIC } [, ...]
```

Parameters

| { EXECUTE | ALL } | The type of privilege to grant the procedure. Either EXECUTE or ALL are applicable privileges to grant. When using more than one schema, specify the schema that contains the procedure. |
| PRIVILEGES | [Optional] For SQL standard compatibility and is ignored. |
| [database.]schema | Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: |
| procedure | The SQL or User Defined procedure on which to grant the privilege. If using more than one schema, you must specify the schema that contains the procedure. |
| argname | The optional argument name for the procedure. |

If you specify a database, it must be the current database.

```sql
myschema.thisDbobject
```
argtype | The required argument data type or types of the procedure.

{ username | role  
  | PUBLIC }[,...] | The recipient of the procedure privileges, which can be one or more users, one or more roles, or all users and roles (PUBLIC).

- **username**—Indicates a specific user
- **role**—Specifies a particular role
- **PUBLIC**—Indicates that all users and roles have granted privileges to the procedure.

---

**Example**

The following command grants EXECUTE privileges on the tokenize procedure to users Bob and Jules, and to the Operator role:

```sql
=> GRANT EXECUTE ON PROCEDURE tokenize(varchar) TO Bob, Jules, Operator;
```

**See Also**

- REVOKE (Procedure)
- Granting and Revoking Privileges

**GRANT (Resource Pool)**

Grants privileges on one or more resource pools to a database user or role. Once granted usage rights, users can switch to using the resource pool with ALTER USER (*username*) or with SET SESSION RESOURCE POOL.

**Syntax**

```sql
GRANT USAGE
  ON RESOURCE POOL resource-pool [,...]
TO { username | role | PUBLIC }[,...]
```
Parameters

<table>
<thead>
<tr>
<th>resource-pool</th>
<th>The resource pools on which to grant the privilege.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ username</td>
<td>role</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>username</strong>—Indicates one or more user names.</td>
</tr>
<tr>
<td></td>
<td>• <strong>role</strong>—Indicates one or more roles.</td>
</tr>
<tr>
<td></td>
<td>• PUBLIC— Indicates that all users and roles have granted privileges to the procedure.</td>
</tr>
</tbody>
</table>

Examples

This examples shows how to grant user Joe usage on resource pool Joe_pool.

```sql
=> CREATE USER Joe;
CREATE USER
=> CREATE RESOURCE POOL Joe_pool;
CREATE RESOURCE POOL
=> GRANT USAGE ON RESOURCE POOL Joe_pool TO Joe;
GRANT PRIVILEGE
```

See Also

- **REVOKE (Resource Pool)**
- **Granting and Revoking Privileges**

**GRANT (Role)**

Adds a predefined role to users or other roles. Granting a role does not activate the role automatically; the user must enable it using the **SET ROLE** command.

Granting a privilege to a role immediately affects active user sessions. When you grant a new privilege, it becomes immediately available to every user with the role active.
Syntax

```
GRANT role [, ...] TO { user | role | PUBLIC } [, ...]
... [ WITH ADMIN OPTION ];
```

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>role [, ...]</code></td>
<td>The name of one or more roles to be granted to users or roles</td>
</tr>
<tr>
<td>`user</td>
<td>role</td>
</tr>
<tr>
<td></td>
<td>keyword PUBLIC is supplied, then all users have access to the role.</td>
</tr>
<tr>
<td><code>WITH ADMIN OPTION</code></td>
<td>Grants users and roles administrative privileges for the role. They are able to grant the role to and revoke the role from other users or roles.</td>
</tr>
</tbody>
</table>

Notes

Vertica will return a NOTICE if you grant a role with or without admin option, to a grantee who has already been granted that role. For example:

```sql
=> GRANT commentor to Bob;
NOTICE 4622: Role "commentor" was already granted to user "Bob"
```

Creating Roles

These examples create three roles, appdata, applogs, and appadmin, and grant one of the roles to a user, bob:

```sql
=> CREATE ROLE appdata;
CREATE ROLE
=> CREATE ROLE applogs;
CREATE ROLE
=> CREATE ROLE appadmin;
CREATE ROLE
=> GRANT appdata TO bob;
GRANT ROLE
```
Activating a Role

After granting a role to a user, the role must be activated. You can activate a role on a session basis, or as part of the user's login.

To activate a role for a user's session:

=> CREATE ROLE appdata;
CREATE ROLE
=> GRANT appdata TO bob;
GRANT ROLE
=> SET ROLE appdata;
SET ROLE

To activate a role as part of the user's login:

=> CREATE ROLE appdata;
CREATE ROLE
=> GRANT appdata TO bob;
GRANT ROLE
=> ALTER USER bob DEFAULT ROLE appdata;
ALTER USER

Granting One Role To Another

Grant two roles to another role:

=> GRANT appdata, applogs TO appadmin;
-- grant to other roles
GRANT ROLE

Now, any privileges assigned to either appdata or applogs are automatically assigned to appadmin as well.

Checking for Circular References

When you grant one role to another role, Vertica combines the newly granted role's permissions with the existing role's permissions. Vertica also checks for circular references when you grant one role to another. The GRANT ROLE function fails with an error if a circular reference is found.

=> GRANT appadmin TO appdata;
WARNING: Circular assignation of roles is not allowed
HINT: Cannot grant appadmin to appdata
GRANT ROLE
Granting Administrative Privileges

A superuser can assign a user or role administrative access to a role by supplying the optional WITH ADMIN OPTION argument to the GRANT statement. Administrative access allows the user to grant and revoke access to the role for other users (including granting them administrative access). Giving users the ability to grant roles lets a superuser delegate role administration to other users.

Note: A user with a DBADMIN role must have the ADMIN OPTION enabled to be able to grant a DBADMIN role to another user.

As with all user privilege models, database superusers should be cautious when granting any user a role with administrative privileges. For example, if the database superuser grants two users a role with administrative privileges, both users can revoke the role of the other user. This example shows granting the appadmin role (with administrative privileges) to users bob and alice. After each user has been granted the appadmin role, either use can connect as the other will full privileges.

```sql
=> GRANT appadmin TO bob, alice WITH ADMIN OPTION;
GRANT ROLE
=> \connect - bob
You are now connected as user "bob".
=> REVOKE appadmin FROM alice;
REVOKE ROLE
```

See Also

- REVOKE (Role)
- Granting and Revoking Privileges

GRANT (Schema)

Grants privileges on a schema to a database user or role.

Note: If a schema was created with Inherited Privileges enabled, any privileges you grant the schema are inherited by all the objects in the table. Otherwise, you need to grant privileges on each object in the table. For more information see CREATE SCHEMA
New users do not have access to schema PUBLIC by default. You must grant USAGE on the PUBLIC schema to all users you create.

**Syntax**

```
GRANT { ... { CREATE | USAGE } [ , ... ] | ALL [ PRIVILEGES ] }
... | { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [ ,... ] ... }
... ON SCHEMA [db-name.]schema [ , ... ]
... TO { username | role | PUBLIC } [ , ... ]
... [ WITH GRANT OPTION ]
```

**Parameters**

<table>
<thead>
<tr>
<th>CREATE</th>
<th>Grants the user read access to the schema and the right to create tables and views within the schema.</th>
</tr>
</thead>
<tbody>
<tr>
<td>USAGE</td>
<td>Grants the user access to the objects contained within the schema. This allows the user to look up objects within the schema. Note that the user must also be granted access to the individual objects. See the GRANT TABLE and GRANT VIEW statements.</td>
</tr>
<tr>
<td>SELECT</td>
<td>With Inherited Privileges enabled on the schema, grants the user SELECT privileges on any column of any table in the schema. See Inherit Privileges on a Schema.</td>
</tr>
<tr>
<td>INSERT</td>
<td>With Inherited Privileges enabled on the schema, grants the user privileges to INSERT tuples into tables in the schema and use the COPY command to load data into the tables. See Inherit Privileges on a Schema.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>With Inherited Privileges enabled on the schema, grants the user privileges to UPDATE tuples in a schema table. See Inherit Privileges on a Schema.</td>
</tr>
<tr>
<td>DELETE</td>
<td>With Inherited Privileges enabled on the schema, grants the user privileges to DELETE rows from a schema table. See Inherit Privileges on a Schema.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>With Inherited Privileges enabled on the schema, grants the ability to create a foreign key constraint. You must have this privilege on both the referencing and referenced tables. You also need USAGE on the schema that contains the table. See Inherit Privileges on a Schema.</td>
</tr>
</tbody>
</table>
With Inherited Privileges enabled on the schema, grants the user TRUNCATE privileges on rows from a schema table. See Inherit Privileges on a Schema.

| ALL            | Grants the user CREATE and USAGE privileges on the schema. |
| PRIVILEGES     | Used for SQL standard compatibility.                     |
| [db-name.]     | [Optional] Specifies the current database name. Using a database name prefix is optional, and does not affect the command in any way. You must be connected to the specified database. |
| schema         | Identifies the schema to which you are granting privileges. |
| username       | Grants the privilege to a specific user.                  |
| role           | Grants the privilege to a specific role.                   |
| PUBLIC         | Grants the privilege to all users.                         |
| WITH GRANT OPTION | Allows the recipient of the privilege to grant it to other users. |

**Examples**

This example shows how to grant user Joe usage on schema online_sales.

```sql
=> CREATE USER Joe;
CREATE USER
=> GRANT USAGE ON SCHEMA online_sales TO Joe;
GRANT PRIVILEGE
```

**See Also**

- REVOKE (Schema)
- Granting and Revoking Privileges

**GRANT (Sequence)**

Grants privileges on a sequence generator to a user or role. Optionally grants privileges on all sequences within one or more schemas.
Syntax

GRANT { SELECT | ALL [ PRIVILEGES ] }... ON SEQUENCE [[database.]schema.]sequence-name[,...]
... | ON ALL SEQUENCES IN SCHEMA schema-name[,...]
... TO { username | role | PUBLIC }[,...]
... [ WITH GRANT OPTION ]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Allows the right to use both the CURRVAL and NEXTVAL functions on the specified sequence.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td>myschema.thisDbObject</td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>sequence-name</td>
<td>Specifies the sequence on which to grant the privileges. When using more than one schema, specify the schema that contains the sequence on which to grant privileges.</td>
</tr>
<tr>
<td>ON ALL SEQUENCES</td>
<td>Grants privileges on all sequences within one or more schemas to a user and/or role.</td>
</tr>
<tr>
<td>IN SCHEMA</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>Grants the privilege to the specified user.</td>
</tr>
<tr>
<td>role</td>
<td>Grants the privilege to the specified role.</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Grants the privilege to all users.</td>
</tr>
<tr>
<td>WITH GRANT OPTION</td>
<td>Allows the user to grant the same privileges to other users.</td>
</tr>
</tbody>
</table>

Privileges

**USAGE on the schema** that contains the sequence
Examples

This example shows how to grant user Joe all privileges on sequence my_seq.

```sql
=> CREATE SEQUENCE my_seq START 100;
CREATE SEQUENCE
=> GRANT ALL PRIVILEGES ON SEQUENCE my_seq TO Joe;
GRANT PRIVILEGE
```

See Also

- `REVOKE (Sequence)`
- `Granting and Revoking Privileges`

**GRANT (Storage Location)**

Grants privileges to non-superusers or roles to read from or write to a Vertica storage location. First, a superuser creates a special class of storage location with the USER keyword through the usage parameter. Creating a storage location with a USER type specifies that the location can be made accessible to non-dbadmin users. The superuser must then grant users or roles the appropriate privileges through the `GRANT (Storage Location)` statement.

**Note:** `GRANT/REVOKE (Storage Location)` statements are applicable only to `USER` storage locations. If the storage location is dropped, all privileges are revoked automatically.

**Syntax**

```
GRANT { READ | WRITE | ALL [ PRIVILEGES ] } ... ON LOCATION 'path' [ ON node ] ... TO { username | role | PUBLIC } [, ...] ... [ WITH Grant OPTION ]
```

**Parameters**

<table>
<thead>
<tr>
<th>READ</th>
<th>Permits the grantee to copy data from files in the storage location into a table.</th>
</tr>
</thead>
</table>
### WRITE
Permits the grantee to export data from the database to the storage location. With WRITE privileges, grantees can also save COPY statement rejected data and exceptions files to the storage location.

### ALL
Grants all available privileges to the grantee for the storage location.

### PRIVILEGES
[Optional] For SQL standard compatibility and is ignored.

### ON LOCATION 'path' [ ON node ]
- **path** — Specifies the path name mount point of the storage location
- **node** — [Optional] Grants access to the storage location residing on the node. If you leave this blank, *node* defaults to all nodes on the specified path in that cluster. If a path exists for only some nodes, the entire grant rolls back, even on the nodes that reside in the path.

### { username | role | PUBLIC } [, ...]
Specifies the privilege grantee, which can be one or more users, one or more roles, or all users (PUBLIC).
- **username** — A specific user
- **role** — A particular role
- PUBLIC — Grants the specified privileges to all users and roles.

### WITH GRANT OPTION
[Optional] Allows the grantee to grant the same privileges to others.

### Notes
Only a superuser can add, alter, retire, drop, and restore a location. The superuser can grant only READ and/or WRITE access privileges to storage locations for other users or roles.
Examples

In the following series of commands, a superuser creates a new storage location and grants it to user Bob:

```sql
=> CREATE LOCATION '/home/dbadmin/UserStorage/BobStore' NODE 'v_mcdb_node0007' USAGE 'USER';
CREATE LOCATION
```

Now the superuser grants a user named Bob all available privileges to the /BobStore location:

```sql
=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' TO Bob;
GRANT PRIVILEGE
```

Revoke all storage location privileges from Bob:

```sql
=> REVOKE ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' FROM Bob;
REVOKE PRIVILEGE
```

Grant privileges to Bob on the BobStore location again, specifying a node:

```sql
=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' ON v_mcdb_node0007 TO Bob;
GRANT PRIVILEGE
```

Revoke all storage location privileges from Bob:

```sql
=> REVOKE ALL ON LOCATION '/home/dbadmin/UserStorage/BobStore' ON v_mcdb_node0007 FROM Bob;
REVOKE PRIVILEGE
```

See Also

- Storage Management Functions
- REVOKE (Storage Location)
- Granting and Revoking Privileges

GRANT (Table)

Grants privileges on a table to a user or role. Optionally grants privileges on all tables within one or more schemas.
Note: Granting privileges on all tables in a schema also includes privileges on all views in the same schema.

Syntax

```sql
GRANT { privilege[, ...] | ALL [ PRIVILEGES ] }
... ON {
... [ TABLE ] [[database.]schema.]table [, ... ]
... | ALL TABLES IN SCHEMA [database.]schema [, ... ]
... TO { username | role | PUBLIC } [, ... ]
... [ WITH GRANT OPTION ]
```

Parameters

<table>
<thead>
<tr>
<th><code>privilege</code></th>
<th>One of the following privileges:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Important: Only SELECT privileges can be granted on system tables.</td>
</tr>
<tr>
<td></td>
<td>• SELECT: Query tables of this schema. SELECT privileges are granted by default to the PUBLIC role.</td>
</tr>
<tr>
<td></td>
<td>• INSERT: Insert rows into tables of this schema, or and load data into tables with COPY.</td>
</tr>
<tr>
<td></td>
<td>Note: COPY FROM STDIN is allowed for users with INSERT privileges, while COPY FROM file requires admin privileges.</td>
</tr>
<tr>
<td></td>
<td>• UPDATE: Update rows of tables of this schema.</td>
</tr>
<tr>
<td></td>
<td>• DELETE: Delete rows of tables of this schema.</td>
</tr>
<tr>
<td></td>
<td>• REFERENCES: Create foreign key constraints for tables of this schema. This privilege must be set on both referencing and referenced tables.</td>
</tr>
<tr>
<td></td>
<td>• TRUNCATE: Truncate table contents. Non-owners of the tables can also execute the following partition operations on them:</td>
</tr>
<tr>
<td>Action/Keyword</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DROP_PARTITIONS</td>
<td>Removes partitions from a table.</td>
</tr>
<tr>
<td>SWAP_PARTITIONS_BETWEEN_TABLES</td>
<td>Exchanges partitions between two tables.</td>
</tr>
<tr>
<td>MOVE_PARTITIONS_TO_TABLE</td>
<td>Moves partitions to a different table.</td>
</tr>
</tbody>
</table>

**ALL [PRIVILEGES]**

Grants all **table privileges** that also belong to the grantor. Grantors cannot grant privileges that they themselves lack.

The optional keyword PRIVILEGES conforms with the SQL standard.

**[database.]schema**

Specifies a schema, by default public. If **schema** is any schema other than public, you must supply the schema name. For example:

| myschema.thisDbObject |

One exception applies: you can specify system tables without their schema name.

If you specify a database, it must be the current database.

**table**

The table on which to grant the privileges.

*Note: The table can be a global temporary table, but not a local temporary table. See [Creating Temporary Tables](#) in the Administrator's Guide.*

**ON ALL TABLES IN SCHEMA**

Grants privileges on all tables (and by default all views) within one or more schemas to a user and/or role.

**username**

Grants the privilege to the specified user.

**role**

Grants the privilege to the specified role.

**PUBLIC**

Grants the privilege to all users.

**WITH GRANT OPTION**

Allows the user to grant the same privileges to other users.
Notes

- The user must also be granted USAGE on the schema that contains the table. See `GRANT (Schema)`.

- To use the `DELETE` or `UPDATE` commands with a `WHERE Clause`, a user must have both `SELECT` and `UPDATE` and `DELETE` privileges on the table.

Examples

Grant user Joe all privileges on table `customer_dimension`:

```
=> CREATE USER Joe;
CREATE USER
=> GRANT ALL PRIVILEGES ON TABLE customer_dimension TO Joe;
GRANT PRIVILEGE
```

Grant user Joe `SELECT` privileges on all system tables:

```
=> GRANT SELECT ON ALL TABLES IN SCHEMA V_MONITOR, V_CATALOG TO Joe;
GRANT PRIVILEGE
```

See Also

- `REVOKE (Table)`

- `Granting and Revoking Privileges`

**GRANT (User Defined Extension)**

Grants privileges on a user-defined extension (UDx) to a database user or role. Optionally grants all privileges on the user-defined extension within one or more schemas. You can grant privileges on the following user-defined extension types:

- User Defined Functions (UDF)
  - User Defined SQL Functions
  - User Defined Scalar Functions (UDSF)
- User Defined Transform Functions (UDTF)
- User Defined Aggregate Functions (UDAF)
- User Defined Analytic Functions (UDAnF)
- User Defined Load Functions (UDL)
  - UDL Filter
  - UDL Parser
  - UDL Source

**Syntax**

```sql
GRANT { EXECUTE | ALL }
... ON FUNCTION [[database.]schema.]function [,]...
... | ON AGGREGATE FUNCTION [[database.]schema.]function [,]...
... | ON ANALYTIC FUNCTION [[database.]schema.]function [,]...
... | ON TRANSFORM FUNCTION [[database.]schema.]function [,]...
... | ON FILTER [[database.]schema.]filter [,]...
... | ON PARSER [[database.]schema.]parser [,]...
... | ON SOURCE [[database.]schema.]source [,]...
... | ON ALL FUNCTIONS IN SCHEMA schema [,]...
... ( [ argname ] argtype [,]... )
... TO { username | role | PUBLIC } [,]...
```

**Parameters**

| { EXECUTE | ALL } | The type of privilege to grant the UDX: |
|----------------|---------------------------------------|
|                | - EXECUTE grants permission to call a user-defined extension. |
|                | - ALL grants all privileges on the user-defined extension |

<table>
<thead>
<tr>
<th>database.</th>
<th>schema</th>
<th>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
</table>
|           | [myschema.]thisDbObject | If you specify a database, it must be the current
| function | The name of the UDX on which to grant the privilege. If you use more than one schema, you must specify the schema that contains the UDX, as noted in the previous row. |
| filter parser source | ON ALL FUNCTIONS IN SCHEMA Grants privileges on all UDX's within one or more schemas to a user, role, or all users and roles. |
| argname | [Optional] The argument name for the UDX. When you GRANT, REVOKE, or DROP privileges for a polymorphic function, you must include an argument with the command. |
| argtype | The argument data type of the UDX. |
| { username | role | PUBLIC } [,...] | The recipient of the UDX privileges, which can be one or more users, one or more roles, or all users and roles (PUBLIC). |
|  | • username - Indicates a specific user |
|  | • role - Specifies a particular role |
|  | • PUBLIC - Indicates that all users and roles have granted privileges to the UDX. |

**Privileges**

Only a superuser and owner can grant privileges on a UDX. To grant privileges to a specific schema UDX or to all UDX's within one or more schemas, grantees must have USAGE privileges on the schema. See GRANT (Schema).

**Examples**

The following command grants EXECUTE privileges on the `myzeroifnull` SQL function to users Bob and Jules, and to the Operator role. The function takes one integer argument:

```sql
=> GRANT EXECUTE ON FUNCTION myzeroifnull (x INT) TO Bob, Jules, Operator;
```
The following command grants EXECUTE privileges on all functions in the zero-schema schema to user Bob:

```sql
=> GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA zero-schema TO Bob;
```

The following command grants EXECUTE privileges on the `tokenize` transform function to user Bob and to the Operator role:

```sql
=> GRANT EXECUTE ON TRANSFORM FUNCTION tokenize(VARCHAR) TO Bob, Operator;
```

The following command grants EXECUTE privileges on the `ExampleSource()` source to user Alice.

```sql
=> CREATE USER Alice;
=> GRANT USAGE ON SCHEMA hdfs TO Alice;
=> GRANT EXECUTE ON SOURCE ExampleSource() TO Alice;
```

The next command grants ALL privileges on the `ExampleSource()` source to user Alice:

```sql
=> GRANT ALL ON SOURCE ExampleSource() TO Alice;
```

The following command grants ALL privileges on the polymorphic function `Pagerank` to the `dbadmin` role:

```sql
=> GRANT ALL ON TRANSFORM FUNCTION Pagerank(z varchar) to dbadmin;
```

See Also

- [REVOKE (User Defined Extension)](REVOKE_UserDefinedExtension.html)
- [Granting and Revoking Privileges in the Administrator's Guide](GrantingAndRevokingPrivileges.html)
- [Developing User-Defined Extensions (UDxs) in Extending Vertica](ExtendingVertica.html)

**GRANT (View)**

Grants privileges on a view to a database user or role.

**Syntax**

```sql
GRANT ... { SELECT | ALL [ PRIVILEGES ]} 
... ON [[database.]schema.]viewname [, ...] 
... TO { username | role | PUBLIC } [, ...] 
... [ WITH GRANT OPTION ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Grants a user or role SELECT operations to a view and any resources referenced within it.</td>
</tr>
<tr>
<td>ALL</td>
<td>Grants a user or role all privileges to a view, and any resources referenced within it.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>[Optional] For SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td>viewname</td>
<td>Specifies the view on which to grant the privileges. When using more than one schema, specify the schema that contains the view, as noted above.</td>
</tr>
<tr>
<td>username</td>
<td>Grants the privilege to the specified user.</td>
</tr>
<tr>
<td>role</td>
<td>Grants the privilege to the specified role.</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Grants the privilege to all users.</td>
</tr>
<tr>
<td>WITH GRANT OPTION</td>
<td>Permits the user to grant the same privileges to other users.</td>
</tr>
</tbody>
</table>

Examples

This example shows how to grant user Joe all privileges on view ship.

```sql
=> CREATE VIEW ship AS SELECT * FROM public.shipping_dimension;
CREATE VIEW
=> GRANT ALL PRIVILEGES ON ship TO Joe;
GRANT PRIVILEGE
```
See Also

- `REVOKE (View)`
- `Granting and Revoking Privileges`
- `Using Views`
- `Privileges Required for Common Database Operations`

**INSERT**

Inserts values into all projections of the specified table. You must insert one complete tuple at a time. By default, `INSERT` first uses WOS. When WOS is full, the inserted tuple overflows to ROS.

If no projections are associated with the target table, Vertica creates a superprojection to store the inserted values.

`INSERT` works for flex tables as well as regular database tables.

**Syntax**

```
INSERT [ /*+ hint[, hint] */ ] INTO [[database.]schema.]table-name
... [ ( column-list ) ]
... { DEFAULT VALUES | VALUES ( values-list ) | SELECT query-expression }
```

**Parameters**

<table>
<thead>
<tr>
<th>/*+ hint [, hint] */</th>
<th>One or both of the following hints:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• A load hint, one of the following: <code>AUTO</code>, <code>DIRECT</code>, or <code>TRICKLE</code></td>
</tr>
<tr>
<td></td>
<td>• <code>LABEL</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>[database.]schema</code></th>
<th>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</th>
</tr>
</thead>
</table>
If you specify a database, it must be the current database.

**table-name**
The target table. You cannot invoke INSERT on a projection. This can be a flex table.

**column-list**
A comma-delimited list of one or more target columns in this table, listed in any order. VALUES clause values are mapped to columns in the same order. If you omit this list, Vertica maps VALUES clause values to columns according to column order in the table definition.

A list of target columns is invalid with DEFAULT VALUES.

**DEFAULT VALUES**
Fills all columns with their default values as specified in the table definition.

You cannot specify a list of target columns with this option.

**VALUES (values-list)**
A comma-delimited list of one or more values to insert in the target columns, where each value is one of the following:

- **expression** resolves to a value to insert in the target column. The expression must not nest other expressions or include Vertica meta-functions.

- DEFAULT inserts the default value as specified in the table definition.

If no value is supplied for a column, Vertica implicitly adds a DEFAULT value, if defined. Otherwise Vertica inserts a NULL value. If the column is defined as NOT NULL, INSERT returns an error.

**SELECT query-expression**
Specifies a query that returns the rows to insert. Isolation level applies only to the SELECT clauses and works like any query.

**Privileges**

- Table owner or user with GRANT OPTION is grantor
- INSERT privilege on table
- USAGE privilege on schema that contains the table

Restrictions

- Vertica does not support subqueries as the target of an INSERT statement.
- If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements.

  Note: Automatic constraint enforcement requires that you have SELECT privileges on the table containing the constraint.

Examples

```sql
=> INSERT INTO t1 VALUES (101, 102, 103, 104);
=> INSERT INTO customer VALUES (10, 'male', 'DPR', 'MA', 35);
=> INSERT INTO retail.t1 (C0, C1) VALUES (1, 1001);
=> INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```

Vertica does not support subqueries or nested expressions as the target of an INSERT statement. For example, the following query returns an error message:

```sql
=> INSERT INTO t1 (col1, col2) VALUES ('abc', (SELECT mycolumn FROM mytable));
ERROR 4821: Subqueries not allowed in target of insert
```

You can rewrite the above query as follows:

```sql
=> INSERT INTO t1 (col1, col2) (SELECT 'abc', mycolumn FROM mytable);
OUTPUT
-------
0
(1 row)
```

The following example shows how to use INSERT . . . VALUES with flex tables:

```sql
=> CREATE FLEX TABLE flex1();
CREATE TABLE
=> INSERT INTO flex1(a,b) VALUES (1, 'x');
OUTPUT
-------
1
```
The next example shows how to use `INSERT...SELECT` with flex tables:

```sql
=> CREATE FLEX TABLE f1x2();
CREATE TABLE
=> INSERT INTO f2x2(a, b) SELECT a, b, '2016-08-10 11:10' c, 'Hello' d, 3.1415 e, f from f1x1;
OUTPUT
-------
1
(1 row)
=> SELECT MapToString(__raw__) FROM f2x2;
MapToString
-----------------
{
  "a" : "1",
  "b" : "x",
  "c" : "2016-08-10",
  "d" : "Hello",
  "e" : 3.1415,
  "f" : null
}
(1 row)
```

### MERGE

Performs update and insert operations on a target table based on the results of a join with another data set, such as a table or view. The join can match a source row with only one target row; otherwise, Vertica returns an error.

A `MERGE` operation uses WOS by default. If WOS fills up, data overflows to ROS.

For detailed information, see [Merging Table Data](#).

### Syntax

```
MERGE [ /*+ hint[, hint] */ ]
... INTO [[database.]schema.]target-table [ [AS] alias ]
... USING source-dataset
... ON join-condition
... matching-clause[ matching-clause ]
```
Returns

Number of target table rows updated or inserted

Parameters

```sql
/*+ hint[, hint]*/  
One or both of the following hints:
  • A load hint, one of the following: AUTO, DIRECT, or TRICKLE
  • LABEL
```

```
[database.]schema  
Specifies a schema, by default public. If `schema` is any schema other than public, you must supply the schema name. For example:

```sql
myschema.thisDbObject
```

If you specify a database, it must be the current database.

`target-table`  
The table on which to perform update and insert operations. `MERGE` takes an X (exclusive) lock on the target table during the operation.

`source-dataset`  
The data to join to `target-table`, one of the following:

```sql
  • `[[database.]schema.]table [ [AS] alias ]`
  • `[[database.]schema.]view [ [AS] alias ]`
  • `(subquery) sq-alias`
```

The specified data set typically supplies the data used to update the target table and populate new rows. You can specify an external table.
### ON join-condition

The conditions on which to join the target table and source data set.

Tip: The Vertica query optimizer can create an optimized query plan for a MERGE statement only if the target table join column has a unique or primary key constraint. For details, see [MERGE Optimization](#) in the Administrator's Guide.

### matching-clause

One of the following clauses:

- **WHEN MATCHED THEN UPDATE**
- **WHEN NOT MATCHED THEN INSERT**

MERGE supports one instance of each clause, and must include at least one.

### WHEN MATCHED THEN UPDATE

For each **target-table** row that is joined (matched) to **source-dataset**, specifies to update one or more columns:

```
WHEN MATCHED [ AND update-filter ] THEN UPDATE
SET { target-column = expression }[, ...]
```

**update-filter** optionally filters the set of matching rows. The update filter can specify any number of conditions. Vertica evaluates each matching row against this filter, and updates only the rows that evaluate to true. For details, see [Update and Insert Filters](#) in the Administrator's Guide.

The following requirements apply:

- A MERGE statement can contain only one **WHEN MATCHED** clause.

- **target-column** can only specify a column name in the target table. It cannot be qualified with a table name.

For details, see [Merging Table Data](#) in the
### WHEN NOT MATCHED THEN INSERT

For each **source-dataset** row that is not joined (not matched) to **target-table**, specifies to:

- Insert a new row into **target-table**.
- Populate each new row with the values specified in **values-list**.

```
WHEN NOT MATCHED [ AND insert-filter ] THEN INSERT
[ ( column-list ) ] VALUES ( values-list )
```

- **column-list** is a comma-delimited list of one or more target columns in the target table, listed in any order. MERGE maps **column-list** columns to **values-list** values in the same order, and each column-value pair must be compatible. If you omit **column-list**, Vertica maps **values-list** values to columns according to column order in the table definition.

- **insert-filter** optionally filters the set of non-matching rows. The insert filter can specify any number of conditions. Vertica evaluates each non-matching source row against this filter. For each row that evaluates to true, Vertica inserts a new row in the target table. For details, see [Update and Insert Filters](#) in the Administrator's Guide.

The following requirements apply:

- A MERGE statement can contain only one **WHEN NOT MATCHED** clause.
- **column-list** can only specify column names in the target table. It cannot be qualified with a table name.
- Insert filter conditions can only reference the source data. If any condition references the target table, Vertica returns an error.

For details, see [Merging Table Data](#) in the Administrator's Guide.
Privileges

MERGE requires the following privileges:

- SELECT permissions on the source data and INSERT, UPDATE, and DELETE permissions on the target table.

- Automatic constraint enforcement requires SELECT permissions on the table containing the constraint.

- SELECT permissions on the target table if the condition in the syntax reads data from the target table. The following example grants user1 access to target table t2:

For example, the following GRANT statement grants user1 access to target table t2. This allows user1 to run the MERGE statement that follows:

```sql
=> GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE t2 to user1;
GRANT PRIVILEGE
=>
```

You are now connected as user "user1".

```sql
=> MERGE INTO t2 USING t1 ON t1.a = t2.a
  WHEN MATCHED THEN UPDATE SET b = t1.b
  WHEN NOT MATCHED THEN INSERT (a, b) VALUES (t1.a, t1.b);
```

Improving MERGE Performance

You can improve MERGE performance in several ways:

- Design projections for optimal MERGE performance.

- Facilitate creation of optimized query plans.

- Use a source data set that is smaller than the target table.

For details, see MERGE Optimization in the Administrator's Guide.
Constraint Enforcement

MERGE respects all enforced constraints in the target table. If the merge operation attempts to copy values that violate those constraints, MERGE returns with an error and rolls back the merge operation.

Caution: If you run MERGE multiple times using the same target and source table, each iteration is liable to introduce duplicate values into the target columns and return with an error.

Columns Prohibited from Merge

The following columns cannot be specified in a merge operation; attempts to do so return with an error:

- Identity/auto-increment columns, or columns whose default value is set to a named sequence.
- Vmap columns such as __raw__ in flex tables.

Examples

See Basic MERGE Example in the Administrator's Guide.

PROFILE

Profiles a single SQL statement.

Syntax

PROFILE { sql-statement }
Parameters

| sql-statement | A query (SELECT) statement or DML statement—for example, you can profile INSERT, UPDATE, COPY, and MERGE. |

Output

Writes profile summary to stderr, saves details to system catalog V_MONITOR.EXECUTION_ENGINE_PROFILES.

Privileges

The same privileges required to run the profiled statement

Description

PROFILE generates detailed information about how the target statement executes, and saves that information in the system catalog V_MONITOR.EXECUTION_ENGINE_PROFILES. Query output is preceded by a profile summary: profile identifiers transaction_id and statement_id, initiator memory for the query, and total memory required. For example:

```sql
=> PROFILE SELECT customer_name, annual_income FROM public.customer_dimension WHERE (customer_gender, annual_income) IN (SELECT customer_gender, MAX(annual_income) FROM public.customer_dimension GROUP BY customer_gender);
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996274683334 and statement_id=7;
NOTICE 3557: Initiator memory for query: [on pool general: 708421 KB, minimum: 554324 KB]
NOTICE 5077: Total memory required by query: [708421 KB]
+----------+-----------------
| customer_name | annual_income |
|------------+--------------|
| Emily G. Vogel | 999998 |
| James M. McNulty | 999979 |
(2 rows)
```

Use profile identifiers to query the table for profile information on a given query.
See Also

Profiling Single Statements

**RELEASE SAVEPOINT**

Destroys a savepoint without undoing the effects of commands executed after the savepoint was established.

**Syntax**

```
RELEASE [ SAVEPOINT ] savepoint_name
```

**Parameters**

- `savepoint_name`  
  Specifies the name of the savepoint to destroy.

**Privileges**

No special permissions required.

**Notes**

Once destroyed, the savepoint is unavailable as a rollback point.

**Example**

The following example establishes and then destroys a savepoint called `my_savepoint`. The values 101 and 102 are both inserted at commit.

```
=> INSERT INTO product_key VALUES (101);
=> SAVEPOINT my_savepoint;
```
See Also

- **SAVEPOINT**
- **ROLLBACK TO SAVEPOINT**
REVOKE Statements

REVOKE statements let you revoke privileges on database objects from users and roles.

REVOKE (Authentication)

Revotes an authentication method that you have granted to (associated with) one or more users or roles.

Syntax

REVOKE AUTHENTICATION auth-method-name FROM
   { Public | user_or_role | user_or_role1, user_or_role2, user_or_role3, ... }

Parameters

<table>
<thead>
<tr>
<th>auth_method_name</th>
<th>Name of the authentication method which you want to revoke from one or more users.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type: VARCHAR</td>
</tr>
<tr>
<td>user_or_role, user_or_role1, user_or_role2, user_or_role3, ...</td>
<td>Names of users or user roles from whom you want to revoke the authentication method.</td>
</tr>
<tr>
<td></td>
<td>Type: VARCHAR</td>
</tr>
</tbody>
</table>

Privileges

Must have DBADMIN privileges.
Examples

This example revokes v_ldap authentication from user jsmith:

```sql
=> REVOKE AUTHENTICATION v_ldap FROM jsmith;
```

This example revokes v_gss authentication from the role DBprogrammer:

```sql
=> REVOKE AUTHENTICATION v_gss FROM DBprogrammer;
```

This example removes localpwd as the default client authentication method:

```sql
=> REVOKE AUTHENTICATION localpwd from Public;
```

See Also

- ALTER AUTHENTICATION
- CREATE AUTHENTICATION
- DROP AUTHENTICATION
- GRANT (Authentication)

REVOKE (Database)

Revokes the right for the specified user or role to create schemas in the specified database.

Syntax

```sql
REVOKE [ GRANT OPTION FOR ]
... { CREATE | TEMP [ , ... ] }
... | CONNECT
... | ALL [ PRIVILEGES ]
... ON DATABASE database-name [ , ... ]
... FROM { username | role }[ , ... ]
... [ CASCADE ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT OPTION FOR</td>
<td>Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Revokes the right to create schemas in the specified database.</td>
</tr>
<tr>
<td>TEMP</td>
<td>Revokes the right to create temp tables in the database.</td>
</tr>
<tr>
<td>ALL</td>
<td>Applies to all privileges.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>database-name</td>
<td>Identifies the database from which to revoke the privilege.</td>
</tr>
<tr>
<td>username</td>
<td>Identifies the user from whom to revoke the privilege.</td>
</tr>
<tr>
<td>role</td>
<td>Identifies the role from which to revoke the privilege.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

Examples

The following example revokes Fred's right to create schemas on vmartdb:

```sql
=> REVOKE CREATE ON DATABASE vmartdb FROM Fred;
```

The following revokes Fred's right to create temporary tables in vmartdb:

```sql
=> REVOKE TEMP ON DATABASE vmartdb FROM Fred;
```

See Also

- GRANT (Database)
- Granting and Revoking Privileges
REVOKE (Library)

Revokes the USAGE privilege on a library from a user or role.

To revoke functions inside the library, a user must have separate REVOKE privileges for those functions.

Syntax

REVOKE { USAGE | ALL }  
... ON LIBRARY [ [ db-name.]schema.]library-name [ , ... ]  
... FROM { username | PUBLIC | role } [ , ... ]  
...[ CASCADE ]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[[db-name.]schema.]</td>
<td>Specifies a schema and an optional database to which you are connected. Include the schema name if multiple schemas exist in the database.</td>
</tr>
<tr>
<td>library-name</td>
<td>The library from which to revoke the USAGE privilege. When using more than one schema, specify the schema that contains the procedure.</td>
</tr>
<tr>
<td>username</td>
<td>role</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Reverses the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>
Privileges

You must have DBADMIN privileges to run REVOKE (Library).

Examples

This example revokes user1's USAGE privilege on the idolLib library in the v_idol schema.

```sql
=> REVOKE USAGE ON LIBRARY v_idol.IdolLib FROM user1;
```

See Also

- GRANT (Library)
- Granting and Revoking Privileges

REVOKE (Model)

Revolves privileges on a model from a user or role.

Syntax

```
REVOKE { ALL | USAGE }
... ON MODEL [ [ db-name.]schema.]modelname [ , ... ]
... FROM { username | role | PUBLIC} [ , ... ]
... [ CASCADE ]
```

Parameters

```
[[db-name.]schema]  Specifies a schema. If multiple schemas are defined in the database,
```
include the schema name. For example:

```sql
myschema
```

### `modelName`

Specifies the model from which to remove privileges.

### `username | role | PUBLIC`

The name of the user, role or group to be granted access.

### `CASCADE`

Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The **CASCADE** keyword first revokes
the privilege from the initial user or role, and then from other grantees extended the privilege.

Example

This example revokes user1's USAGE privilege on the mySvmClassModel model:

```sql
=> REVOKE USAGE ON mySvmClassModel FROM user1;
```

See Also

- GRANT (Model)
- Managing Model Security

REVOKE (Procedure)

Revolves the execute privilege on a procedure from a user or role.

Syntax

```
REVOKE EXECUTE
... ON PROCEDURE [[database.]schema.]procedure[,...]
... ( [ argname ] argtype[,.... ] )
... FROM { PUBLIC | { username | role }[,...] }
...[ CASCADE ]
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example: myschema.thisDbObject</td>
</tr>
<tr>
<td>procedure</td>
<td>Specifies the procedure on which to revoke the execute privilege.</td>
</tr>
<tr>
<td>argname</td>
<td>Optionally specifies the argument names used when creating the procedure.</td>
</tr>
<tr>
<td>argtype</td>
<td>Specifies the argument types used when creating the procedure.</td>
</tr>
<tr>
<td>username</td>
<td>Specifies the user from whom to revoke the privilege.</td>
</tr>
<tr>
<td>role</td>
<td>Specifies the role from whom to revoke the privilege.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

Privileges

You must have DBADMIN privileges to run REVOKE (Procedure).

Examples

This example revokes Bob's execute privilege on the tokenize procedure.

```sql
=> REVOKE EXECUTE ON PROCEDURE tokenize(varchar) FROM Bob;
```
See Also

- GRANT (Procedure)
- Granting and Revoking Privileges

REVOKE (Resource Pool)

Revolves a user's or role's access privilege to a resource pool.

Syntax

REVOKE USAGE... ON RESOURCE POOL resource-pool
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]

Parameters

<table>
<thead>
<tr>
<th>resource-pool</th>
<th>Specifies the resource pool from which to revoke the usage privilege.</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Revokes the privilege from the specified user.</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Revokes the privilege from all users.</td>
</tr>
<tr>
<td>role</td>
<td>Revokes the privilege from the specified role.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

Notes

- Vertica checks privileges on resource pools during runtime. If the user running a query does not have the USAGE privilege on the appropriate pool, the query fails with an error. In this case, the user can use ALTER USER ... RESOURCE POOL to access another resource pool.
Note the user may need to be granted usage on the resource pool with GRANT USAGE ON RESOURCE POOL.

- Revoking a user's permission from a resource pool in another session affects the user's access current session. In this case, the current session fails with an error stating that the resource pool in the current session does not exist.

**Examples**

This example shows how to revoke user Joe's usage privilege on the Joe_pool resource pool.

```sql
=> REVOKE USAGE ON RESOURCE POOL Joe_pool FROM Joe;
REVOKE PRIVILEGE
```

**See Also**

- [GRANT (Resource Pool)]
- [Granting and Revoking Privileges]

**REVOKE (Role)**

Revokes a role (and administrative access, applicable) from a grantee. A user that has administrator access to a role can revoke the role for other users.

You can also remove a role's access to another role.

**Syntax**

```sql
REVOKE [ ADMIN OPTION FOR ] role [, ...] 
[ ... FROM { user | role | PUBLIC } [, ...] 
[ CASCADE ];
```

**Parameters**

<table>
<thead>
<tr>
<th>ADMIN OPTION FOR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Revokes just the user's or role's administration access to the role, and not the role itself.</td>
</tr>
</tbody>
</table>
role
The name of one or more roles from which you want to revoke access.

user | role | PUBLIC
The name of a user or role whose permission you want to revoke. You can use the PUBLIC option to revoke access to a role that was previously made public.

CASCADE
Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.

Examples
This example shows the revocation of the pseudosuperuser role from the dbadmin user:

```sql
=> REVOKE pseudosuperuser from dbadmin;
```

This example shows the revocation of administration access from the dbadmin user for the pseudosuperuser role. The ADMIN OPTION command does not remove the pseudosuperuser role.

```sql
=> REVOKE ADMIN OPTION FOR pseudosuperuser FROM dbadmin;
```

Notes
If the role you are trying to revoke was not already granted to the user, Vertica returns a NOTICE:

```sql
=> REVOKE commentor FROM Sue;
NOTICE 2022: Role "commentor" was not already granted to user "Sue"
REVOKE ROLE
```

See Also
- GRANT (Role)
- Granting and Revoking Privileges
REVOKE (Schema)

Revolves privileges on a schema from a user or role.

Note: In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

REVOKE [ GRANT OPTION FOR ] { ...
  { CREATE | USAGE } [...]
  { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [...]
  ALL [ PRIVILEGES ]

ON SCHEMA [db-name.] schema [...]

FROM { username | PUBLIC | role } [...]

[ CASCADE ]

Parameters

<table>
<thead>
<tr>
<th>GRANT OPTION FOR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Revokes the right to create tables and views in the schema.</td>
</tr>
<tr>
<td>USAGE</td>
<td>Revokes user access to the objects contained in the schema. Note that the user can also have access to the individual objects revoked. See the GRANT TABLE and GRANT VIEW statements.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Revokes the user's ability to perform a SELECT on any column of any table in the schema.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Revokes the user's ability to INSERT tuples into tables in the schema and use the COPY command to load data into the tables.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Revokes the user's ability to UPDATE tuples in a schema table.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Revokes the user's ability to DELETE rows from a schema table.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Revokes the user's privilege on both the referencing and referenced tables for creating a foreign key constraint.</td>
</tr>
<tr>
<td>ALL</td>
<td>Revokes all privileges previously granted.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>[db-name.]</td>
<td>[Optional] Specifies the current database name. Using a database name prefix is optional, and does not affect the command in any way. You must be connected to the specified database.</td>
</tr>
<tr>
<td>schema</td>
<td>Identifies the schema from which to revoke privileges.</td>
</tr>
<tr>
<td>username</td>
<td>Revokes the privilege to a specific user.</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Revokes the privilege to all users.</td>
</tr>
<tr>
<td>role</td>
<td>Revokes the privilege to a specific role.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows how to revoke user Joe's usage privileges on the online_sales schema.

```sql
=> REVOKE USAGE ON SCHEMA online_sales FROM Joe;
REVOKE PRIVILEGE
```

**See Also**

- [GRANT (Schema)]
- [Granting and Revoking Privileges]

**REVOKE (Sequence)**

Revokes privileges on a sequence generator from a user or role. Optionally revokes privileges on all sequences within one or more schemas.
Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { SELECT | ALL [ PRIVILEGES ] }
... ON SEQUENCE [ [database.]schema.]sequence [ , ... ]
... | ON ALL SEQUENCES IN SCHEMA schema [, ...]
... FROM { username | PUBLIC | role } [ , ... ]
...[ CASCADE ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Revokes the right to use both the CURRVAL and NEXTVAL functions on the specified sequence.</td>
</tr>
<tr>
<td>ALL</td>
<td>Applies to all privileges.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>[database.]schema</td>
<td>Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td>sequence</td>
<td>Specifies the sequence from which to revoke privileges.</td>
</tr>
<tr>
<td>ON ALL SEQUENCES IN SCHEMA</td>
<td>Revokes privileges on all sequences within one or more schemas from a user and/or role.</td>
</tr>
<tr>
<td>username</td>
<td>Revokes the privilege from the specified user.</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>Revokes the privilege from all users.</td>
</tr>
<tr>
<td>role</td>
<td>Revokes the privilege from the specified role.</td>
</tr>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and</td>
</tr>
</tbody>
</table>
then from other grantees extended the privilege.

Examples

This example shows how to revoke user Joe's privileges on the my_seq sequence.

```sql
=> REVOKE ALL PRIVILEGES ON SEQUENCE my_seq FROM Joe;
REVOKE PRIVILEGE
```

See Also

- **GRANT (Sequence)**
- **Granting and Revoking Privileges**

**REVOKE (Storage Location)**

Revokes privileges from a user or role to read from or write to a storage location.

**Note:** The REVOKE (Storage Location) statement is applicable only to 'USER' storage locations. See **GRANT (Storage Location)** for more information. If the storage location is dropped, all user privileges are removed as part of that.

Syntax

```
REVOKE [ GRANT OPTION FOR ]
... { READ | WRITE | ALL [ PRIVILEGES ] }
... ON LOCATION [ 'path' [ ON node ] ]
... FROM { username | role | PUBLIC } [, ...]
... [ CASCADE ]
```

Parameters

<table>
<thead>
<tr>
<th>REVOKE OPTION FOR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REVOKE OPTION FOR</td>
<td>Revokes GRANT privileges from the grantee.</td>
</tr>
<tr>
<td>READ</td>
<td>Revokes privileges from the grantee to copy data from files in a storage locations into a table.</td>
</tr>
</tbody>
</table>
### WRITE

Revoices privileges to export Vertica data from the database to a storage location. Revoking WRITE privileges prevents the grantee from exporting COPY statement rejected data and exceptions files to a storage location.

### ALL

Revoices all privileges from the grantee to the storage location.

### PRIVILEGES

For SQL standard compatibility and is ignored.

```sql
{ username | role | PUBLIC } [...]  
```

Revoices privileges from one or more users, one or more roles, or all users (PUBLIC).

- **username** – A specific user
- **role** – A particular role
- PUBLIC – Revokes privileges from all users and roles.

### ON LOCATION ( ‘path’ [ ON node ] )

- **path** — Specifies the path name mount point of the storage location
- **node** — [Optional] Revokes access to the storage location residing on the node. If you leave this blank, `node` defaults to all nodes on the specified path in that cluster. If a path exists for only some nodes, the entire grant rolls back, even on the nodes that reside in the path.

### CASCADE

Revoices privileges from the grantee. Using the CASCADE keyword when revoking privileges first removes them from the initial grantee. The statement then revokes all privileges from other users and roles to whom the grantee extended storage location access.

## Examples

For examples, see [GRANT (Storage Location)](Vertica Analytic Database (9.0.x))
See Also

- Granting and Revoking Privileges

REVOKE (Table)

Revokes privileges on a table from a user or role. Optionally revokes privileges on all tables within one or more schemas.

Note: Revoking privileges on all tables within a schema includes all views in the same schema.

In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

REVOKE [ GRANT OPTION FOR ] { privilege[,...] | ALL [ PRIVILEGES ] }

... ON {

..... [ TABLE ] [[database.]schema.]tablename [ ,... ]

..... | ON ALL TABLES IN SCHEMA schema-name [,...] }

... FROM { username | PUBLIC | role } [ ,... ]

... [ CASCADE ]

Parameters

<table>
<thead>
<tr>
<th>GRANT OPTION FOR</th>
<th>Revokes the grant option for the privilege, not the privilege itself. If omitted, revokes both the privilege and the grant option.</th>
</tr>
</thead>
<tbody>
<tr>
<td>privilege</td>
<td>One of the following privileges:</td>
</tr>
<tr>
<td></td>
<td><strong>Important</strong>: Only SELECT privileges can be revoked on system tables.</td>
</tr>
<tr>
<td></td>
<td>• SELECT: Query tables of this schema.</td>
</tr>
<tr>
<td></td>
<td>• INSERT: Insert rows into tables of this schema, or and load data into tables with COPY.</td>
</tr>
</tbody>
</table>
Note: COPY FROM STDIN is allowed all users with INSERT privileges, while COPY FROM file requires admin privileges.

- **UPDATE**: Update rows of tables of this schema.
- **DELETE**: Delete rows of tables of this schema.
- **REFERENCES**: Create foreign key constraints for tables of this schema.
- **TRUNCATE**: Truncate table contents. Non-owners of the tables can no longer execute the following partition operations on them:
  - DROP_PARTITIONS
  - SWAP_PARTITIONS_BETWEEN_TABLES
  - MOVE_PARTITIONS_TO_TABLE

### ALL [PRIVILEGES]

Revoke all table privileges. The optional keyword PRIVILEGES is supported to comply with the SQL standard.

### [database.]schema

Specifies a schema, by default public. If schema is any schema other than public, you must supply the schema name. For example:

```sql
myschema.thisDbObject
```

One exception applies: you can specify system tables without their schema name.

If you specify a database, it must be the current database.

### tablename

Specifies the table from which to remove privileges.

### ON ALL TABLES IN SCHEMA

Revoke privileges on all tables (and by default views) within one or more schemas from a user and/or role.

### username

Revoke the privilege from the specified user.

### PUBLIC

Revoke the privilege from all users.
<table>
<thead>
<tr>
<th>role</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CASCADE</td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows how to revoke user Joe's privileges on the customer_dimension table.

```sql
=> REVOKE ALL PRIVILEGES ON TABLE customer_dimension FROM Joe;
REVOKE PRIVILEGE
```

**See Also**

- [GRANT (Table)](#)
- [Granting and Revoking Privileges](#)

**REVOKE (User Defined Extension)**

Revokes the EXECUTE privilege on a user-defined extension (UDx) from a database user or role. Optionally revokes privileges on all user-defined extensions within one or more schemas. You can revoke privileges on the following user-defined extension types:

- User Defined Functions (UDF)
  - User Defined SQL Functions
  - User Defined Scalar Functions (UDSF)
  - User Defined Transform Functions (UDTF)
  - User Defined Aggregate Functions (UDAF)
  - User Defined Analytic Functions (UDAnF)
- **User Defined Load Functions (UDL)**
  - UDL Filter
  - UDL Parser
  - UDL Source

### Syntax

```
REVOKE EXECUTE
... ON FUNCTION [ schema.]function [,..]  
... | ON AGGREGATE FUNCTION [[database.]schema.]function [,..]  
... | ON ANALYTIC FUNCTION [[database.]schema.]function [,..]  
... | ON TRANSFORM FUNCTION [[database.]schema.]function [,..]  
... | ON FILTER [[database.]schema.]filter [,..]  
... | ON PARSER [[database.]schema.]parser[,..]  
... | ON SOURCE [[database.]schema.]source [,..]  
... | ON ALL FUNCTIONS IN SCHEMA schema [,..]  
... ( [ argname ] argtype [,..] )  
... FROM { username | role | PUBLIC } [,..]  
... [ CASCADE ]
```

### Parameters

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td><code>function</code></td>
<td>The name of the U Dx from which to revoke the EXECUTE privilege. If you use more than one schema, you must specify the schema that contains the U D x, as noted in the previous row.</td>
</tr>
<tr>
<td><code>filter</code></td>
<td></td>
</tr>
<tr>
<td><code>parser</code></td>
<td></td>
</tr>
<tr>
<td><code>source</code></td>
<td></td>
</tr>
<tr>
<td><strong>ON ALL FUNCTIONS IN SCHEMA</strong></td>
<td>Revokes EXECUTE privileges on all U Dx's within one or more schemas from a user, role, or all users and roles.</td>
</tr>
<tr>
<td><code>argname</code></td>
<td>[Optional] The argument name or names for the</td>
</tr>
</tbody>
</table>
When you `GRANT`, `REVOKE`, or `DROP` privileges for a polymorphic function, you must include an argument with the command.

<table>
<thead>
<tr>
<th>argtype</th>
<th>The argument data type or types of the UDX.</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{ username</td>
<td>role</td>
</tr>
<tr>
<td><code>CASCADE</code></td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The CASCADE keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

**Privileges**

You must have DBADMIN privileges, or be the owner of the UDX, to run `REVOKE` (User Defined Extension).

**Examples**

The following command revokes EXECUTE privileges from user Bob on the `myzeroifnull` function:

```
> REVOKE EXECUTE ON FUNCTION myzeroifnull (x INT) FROM Bob;
```

The following command revokes ALL privileges from user Doug on the `Pagerank` polymorphic function:

```
> REVOKE ALL ON TRANSFORM FUNCTION Pagerank (t float) FROM Doug;
```

The following command revokes EXECUTE privileges on all functions in the `zero-schema` schema from user Bob:

```
> REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA zero-schema FROM Bob;
```

The following command revokes EXECUTE privileges from user Bob on the `tokenize` function:
> REVOKE EXECUTE ON TRANSFORM FUNCTION tokenize(VARCHAR) FROM Bob;

The following command revokes ALL privileges on the ExampleSource() source from user Alice:

> REVOKE ALL ON SOURCE ExampleSource() FROM Alice;

See Also

- GRANT (User Defined Extension)
- Granting and Revoking Privileges in the Administrator's Guide
- Developing User-Defined Extensions (UDxs) in Extending Vertica

REVOKE (View)

Revoke user privileges on a view.

**Important:** In a database with trust authentication, the GRANT and REVOKE statements appear to work as expected but have no actual effect on the security of the database.

Syntax

```
REVOKE [ GRANT OPTION FOR ]...
{ SELECT | ALL [ PRIVILEGES ]}
... ON [[ database. ]schema. ]viewname [ , ... ]
... FROM { username | PUBLIC } [ , ... ]
... [ CASCADE ]
```

Parameters

<table>
<thead>
<tr>
<th>GRANT OPTION FOR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Revokes the user's ability to perform SELECT operations on a view and its referenced resources.</td>
</tr>
<tr>
<td>ALL</td>
<td>Revokes all previously granted privileges.</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td>Is for SQL standard compatibility and is ignored.</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>viewname</code></td>
<td>Specifies the view on which to revoke the privileges.</td>
</tr>
<tr>
<td><code>username</code></td>
<td>Revokes the privilege from the specified user.</td>
</tr>
<tr>
<td><code>PUBLIC</code></td>
<td>Revokes the privilege from all users.</td>
</tr>
<tr>
<td><code>CASCADE</code></td>
<td>Revokes the privilege from the specified user or role and then from others. After a user or role has been granted a privilege, the user can grant that privilege to other users and roles. The <code>CASCADE</code> keyword first revokes the privilege from the initial user or role, and then from other grantees extended the privilege.</td>
</tr>
</tbody>
</table>

### Examples

This example shows how to revoke SELECT privileges from user Joe on the view named `test_view`.

```
=> REVOKE SELECT ON test_view FROM Joe;
REVOKE PRIVILEGE
```

You can also use the database name and schema name in the command:

```
=> REVOKE SELECT ON VMart.public.test_view FROM Joe;
REVOKE PRIVILEGE
```

### See Also

- [GRANT (View)]
- [Granting and Revoking Privileges]
ROLLBACK

Ends the current transaction and discards all changes that occurred during the transaction.

Syntax

ROLLBACK [ WORK | TRANSACTION ]

Parameters

| WORK | TRANSACTION | Have no effect; they are optional keywords for readability. |

Privileges

No special permissions required.

Notes

When an operation is rolled back, any locks that are acquired by the operation are also rolled back.

ABORT is a synonym for ROLLBACK.

Examples

This example shows how to roll back from a DELETE transaction.

=> SELECT * FROM sample_table;
a
---
1
(1 row)
=> DELETE FROM sample_table WHERE a = 1;
This example shows how to roll back the changes you made since the BEGIN statement.

```
=> SELECT * FROM sample_table;
a
---
(0 rows)
=> ROLLBACK;
=> SELECT * FROM sample_table;
a
---
1
(1 row)
```

See Also

- Transactions
- Creating Transactions
- BEGIN
- COMMIT
- END
- START TRANSACTION

**ROLLBACK TO SAVEPOINT**

Rolls back all commands that have been entered within the transaction since the given savepoint was established.

**Syntax**

```
ROLLBACK TO [SAVEPOINT] savepoint_name
```
Parameters

| savepoint_name | Specifies the name of the savepoint to roll back to. |

Privileges

No special permissions required.

Notes

- The savepoint remains valid and can be rolled back to again later if needed.
- When an operation is rolled back, any locks that are acquired by the operation are also rolled back.
- ROLLBACK TO SAVEPOINT implicitly destroys all savepoints that were established after the named savepoint.

Example

The following example rolls back the values 102 and 103 that were entered after the savepoint, my_savepoint, was established. Only the values 101 and 104 are inserted at commit.

```sql
=> INSERT INTO product_key VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (102);
=> INSERT INTO product_key VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO product_key VALUES (104);
=> COMMIT;
```

See Also

- RELEASE SAVEPOINT
- SAVEPOINT
SAVEPOINT

Creates a special mark, called a savepoint, inside a transaction. A savepoint allows all commands that are executed after it was established to be rolled back, restoring the transaction to the state it was in at the point in which the savepoint was established.

Tip: Savepoints are useful when creating nested transactions. For example, a savepoint could be created at the beginning of a subroutine. That way, the result of the subroutine could be rolled back if necessary.

Syntax

SAVEPOINT savepoint_name

Parameters

| savepoint_name | Specifies the name of the savepoint to create. |

Privileges

No special permissions required.

Notes

- Savepoints are local to a transaction and can only be established when inside a transaction block.
- Multiple savepoints can be defined within a transaction.
- If a savepoint with the same name already exists, it is replaced with the new savepoint.
Example

The following example illustrates how a savepoint determines which values within a transaction can be rolled back. The values 102 and 103 that were entered after the savepoint, my_savepoint, was established are rolled back. Only the values 101 and 104 are inserted at commit.

```sql
=> INSERT INTO T1 (product_key) VALUES (101);
=> SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (102);
=> INSERT INTO T1 (product_key) VALUES (103);
=> ROLLBACK TO SAVEPOINT my_savepoint;
=> INSERT INTO T1 (product_key) VALUES (104);
=> COMMIT;
=> SELECT product_key FROM T1;

101
104
(2 rows)
```

See Also

- RELEASE SAVEPOINT
- ROLLBACK TO SAVEPOINT

SELECT

Retrieves a result set from one or more tables.

Syntax

```sql
[[ AT EPOCH {LATEST | epoch-number} | AT TIME 'timestamp']]
[ AT epoch ]

SELECT [* | /*+ LABEL(label-name)*/ ] [ ALL | DISTINCT ]
... { * | expression [ [AS] output-name] }[,...]
... [ INTO TABLE ]
... [ from-clause ]
... [ WHERE condition ]
... [ TIMESERIES slice-time ]
```
... [ GROUP BY expression[, ...] ]
... [ HAVING condition[, ...] ]
... [ MATCH ]
... [ UNION { ALL | DISTINCT } ]
... [ EXCEPT ]
... [ INTERSECT ]
... [ ORDER BY expression [ ASC | DESC ][, ...] ]
... [ LIMIT { count | ALL } ]
... [ OFFSET start ]
... [ FOR UPDATE [ OF table-name[, ...] ] ]

Parameters

Note: SELECT clauses such as INTO and WHERE are discussed in sub-sections of this page.

<table>
<thead>
<tr>
<th>AT epoch</th>
<th>Returns data from the specified epoch, where epoch is one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• EPOCH LATEST: Return data from the latest committed DML transaction.</td>
</tr>
<tr>
<td></td>
<td>• EPOCH integer: Return data from the integer-specified epoch.</td>
</tr>
<tr>
<td></td>
<td>• TIME 'timestamp': Return data from the timestamp-specified epoch.</td>
</tr>
<tr>
<td></td>
<td>These options are ignored if used to query temporary tables.</td>
</tr>
<tr>
<td></td>
<td>For details, see Historical Queries in Analyzing Data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>/<em>+LABEL (label-name)</em>/</th>
<th>Assigns a label to a query so you can identify it for profiling and debugging.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In a UNION statement, only the first SELECT statement can be labeled; Vertica ignores labels in subsequent SELECT statements.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALL</th>
<th>DISTINCT</th>
<th>ALL (default): Retains duplicate rows in result set or group.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• DISTINCT: Removes duplicate rows from the result set or group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ALL or DISTINCT qualifier must immediately follow the SELECT keyword. Only one instance of this keyword can appear in the select list.</td>
</tr>
</tbody>
</table>

| * | Lists all columns in the queried tables. |
Caution: Selecting all columns from the queried tables can produce a very large wide set, which can adversely affect performance.

(expression [AS output-name])

A table column or column expression to select from the queried tables. You can optionally qualify expression with an output name, which can be used in several ways:

- Label the column for display.
- Refer to the column's value in ORDER BY and GROUP BY clauses (it cannot be referenced in WHERE or HAVING clauses).

from-clause

A comma-separated list of data sources to query.

FOR UPDATE

Specifies to obtain an X lock on all tables specified in the query, most often used from READ COMMITTED isolation.

FOR UPDATE requires update/delete permissions on the queried tables and cannot be issued from a read-only transaction.

Privileges

You must have USAGE privileges on the schemas that contain the queried tables, as well as one, but not both, of the following:

- Owner or user with GRANT OPTION privileges
- SELECT privilege

If the SELECT statement queries a view, the view owner must have SELECT privileges on the view's anchor tables or views.

Example

When multiple clients run transactions as in the following example query, deadlocks can occur if FOR UPDATE is not used. Two transactions acquire an S lock, and when both attempt to upgrade to an X lock, they encounter deadlocks:

```sql
=> SELECT balance FROM accounts WHERE account_id=3476 FOR UPDATE;
...
```
UPDATE accounts SET balance = balance+10 WHERE account_id=3476;
=> COMMIT;

See Also

- LOCKS
- Analytic Functions
- SQL Analytics
- Time Series Analytics
- Event Series Pattern Matching
- Subqueries
- Joins

EXCEPT Clause

Combines two or more SELECT queries. EXCEPT returns distinct results of the left-hand query that are not also found in the right-hand query.

Note: MINUS is an alias for EXCEPT.

Syntax

```sql
SELECT
    ...
    EXCEPT select
    ...
    [ EXCEPT select ]...
    ...
    [ ORDER BY { column-name
    ...
    | ordinal-number }
    ...
    [ ASC | DESC ] [ , ... ] ]
    ...
    [ LIMIT { integer | ALL }]
    ...
    [ OFFSET integer ]
```
Notes

- Use the EXCEPT clause to filter out specific results from a SELECT statement. The EXCEPT query operates on the results of two or more SELECT queries. It returns only those rows in the left-hand query that are not also present in the right-hand query.

- Vertica evaluates multiple EXCEPT clauses in the same SELECT query from left to right, unless parentheses indicate otherwise.

- You cannot use the ALL keyword with an EXCEPT query.

- The results of each SELECT statement must be union compatible. Each statement must return the same number of columns, and the corresponding columns must have compatible data types. For example, you cannot use the EXCEPT clause on a column of type INTEGER and a column of type VARCHAR. If statements do not meet these criteria, Vertica returns an error.

  Note: The Data Type Coercion Chart lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

- You can use EXCEPT in FROM, WHERE, and HAVING clauses.

- You can order the results of an EXCEPT operation by including an ORDER BY operation in the statement. When you write the ORDER BY list, specify the column names from the leftmost SELECT statement, or specify integers that indicate the position of the columns by which to sort.

- The rightmost ORDER BY, LIMIT, or OFFSET clauses in an EXCEPT query do not need to be enclosed in parentheses, because the rightmost query specifies that Vertica perform the operation on the results of the EXCEPT operation. Any ORDER BY, LIMIT, or OFFSET clauses contained in SELECT queries that appear earlier in the EXCEPT query must be enclosed in parentheses.

- Vertica supports EXCEPT noncorrelated subquery predicates. For example:

```sql
=> SELECT * FROM T1
  WHERE T1.x IN
    (SELECT MAX(c1) FROM T2
     EXCEPT
     SELECT MAX(cc1) FROM T3
     EXCEPT
     SELECT MAX(d1) FROM T4);
```
Examples

Consider the following three tables:

**Company_A**

<table>
<thead>
<tr>
<th>Id</th>
<th>emp_lname</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>auto parts</td>
<td>1000</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
<td>auto parts</td>
<td>2500</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>floral</td>
<td>500</td>
</tr>
<tr>
<td>3214</td>
<td>Smithson</td>
<td>sporting goods</td>
<td>1500</td>
</tr>
</tbody>
</table>

(4 rows)

**Company_B**

<table>
<thead>
<tr>
<th>Id</th>
<th>emp_lname</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4321</td>
<td>Marvin</td>
<td>home goods</td>
<td>250</td>
</tr>
<tr>
<td>8765</td>
<td>Bob</td>
<td>electronics</td>
<td>20000</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>home goods</td>
<td>500</td>
</tr>
<tr>
<td>3214</td>
<td>Smithson</td>
<td>home goods</td>
<td>1500</td>
</tr>
</tbody>
</table>

(4 rows)

**Company_C**

<table>
<thead>
<tr>
<th>Id</th>
<th>emp_lname</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>3214</td>
<td>Smithson</td>
<td>sporting goods</td>
<td>1500</td>
</tr>
<tr>
<td>5432</td>
<td>Madison</td>
<td>sporting goods</td>
<td>400</td>
</tr>
<tr>
<td>7865</td>
<td>Cleveland</td>
<td>outdoor</td>
<td>1500</td>
</tr>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>floral</td>
<td>1000</td>
</tr>
</tbody>
</table>

(4 rows)

The following query returns the IDs and last names of employees that exist in Company_A, but not in Company_B:

```sql
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B;
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>emp_lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
</tr>
</tbody>
</table>
```

(2 rows)

The following query sorts the results of the previous query by employee last name:

```sql
=> SELECT id, emp_lname FROM Company_A
      EXCEPT
      SELECT id, emp_lname FROM Company_B
      ORDER BY emp_lname ASC;
```

```sql
<table>
<thead>
<tr>
<th>id</th>
<th>emp_lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
</tr>
</tbody>
</table>
```
If you order by the column position, the query returns the same results:

```sql
=> SELECT id, emp_lname FROM Company_A
   EXCEPT
   SELECT id, emp_lname FROM Company_B
   ORDER BY 2 ASC;
```

id | emp_lname
---+----------------
5678 | Alice
1234 | Stephen
(2 rows)

The following query returns the IDs and last names of employees that exist in Company_A, but not in Company_B or Company_C:

```sql
=> SELECT id, emp_lname FROM Company_A
   EXCEPT
   SELECT id, emp_lname FROM Company_B
   EXCEPT
   SELECT id, emp_lname FROM Company_C;
```

id | emp_lname
---+----------------
5678 | Alice
1234 | Stephen
(1 row)

The following query shows the results of mismatched data types:

```sql
=> SELECT id, emp_lname FROM Company_A
   EXCEPT
   SELECT emp_lname, id FROM Company_B;
ERROR 3429: For 'EXCEPT', types int and varchar are inconsistent
DETAIL:  Columns: id and emp_lname
```

Using the VMart example database, the following query returns information about all Connecticut-based customers who bought items through stores and whose purchases amounted to more than $500, except for those customers who paid cash:

```sql
=> SELECT customer_key, customer_name FROM public.customer_dimension
   WHERE customer_key IN (SELECT customer_key FROM store.store_sales_fact
                             WHERE sales_dollar_amount > 500
                             EXCEPT
                             SELECT customer_key FROM store.store_sales_fact
                             WHERE tender_type = 'Cash')
   AND customer_state = 'CT';
customer_key | customer_name
-------------+----------------
15084 | Doug V. Lampert
21730 | Juanita F. Peterson
### See Also

- SELECT
- INTERSECT Clause
- UNION Clause
- Subqueries

### FROM Clause

A comma-separated list of data sources to query.

### Syntax

```
FROM dataset [, ...] [ TABLESAMPLE(percent) ]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dataset</code></td>
<td>A set of data to query, one of the following:</td>
</tr>
<tr>
<td></td>
<td>- Table reference</td>
</tr>
<tr>
<td></td>
<td>- Joined tables</td>
</tr>
<tr>
<td></td>
<td>- Named subquery:</td>
</tr>
<tr>
<td></td>
<td>* subquery [AS] name</td>
</tr>
<tr>
<td>TABLESAMPLE(percent)</td>
<td>Valid only for user-defined tables, specifies to return a simple random</td>
</tr>
<tr>
<td></td>
<td>sampling of records, where <code>percent</code> specifies the approximate</td>
</tr>
</tbody>
</table>
sampling size. The `percent` value must be > 0 and < 100, and can be a decimal value. The number of records returned is not guaranteed to be the exact percentage specified.

Each row of the data set has the same opportunity to be selected. Vertica performs this sampling before other filters in the query are applied.

This clause is not valid for views, or data collector and system tables.

**Examples**

Return all records from the `customer_dimension` table:

```sql
=> SELECT * FROM customer_dimension;
```

The following example shows how to use the `TABLESAMPLE` clause. In this example, the query returns 50% of the rows in the table:

```sql
=> SELECT * FROM user_name TABLESAMPLE(50);

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Barry</td>
</tr>
<tr>
<td>982</td>
<td>Nero</td>
</tr>
<tr>
<td>8761</td>
<td>Lou</td>
</tr>
</tbody>
</table>

(3 rows)
```

In the next example, the query returns a small percentage of the rows in the `customer_dimension` table:

```sql
=> SELECT customer_name, customer_state FROM customer_dimension TABLESAMPLE(.015);

<table>
<thead>
<tr>
<th>customer_name</th>
<th>customer_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steve X. Dobisz</td>
<td>UT</td>
</tr>
<tr>
<td>Ben C. Kramer</td>
<td>IA</td>
</tr>
<tr>
<td>Alexandra A. Peterson</td>
<td>IL</td>
</tr>
<tr>
<td>Dean B. Nielson</td>
<td>SD</td>
</tr>
<tr>
<td>Sally F. McNulty</td>
<td>CT</td>
</tr>
<tr>
<td>Americare</td>
<td>CA</td>
</tr>
<tr>
<td>Wendy K. Carcetti</td>
<td>NJ</td>
</tr>
<tr>
<td>Steve V. Williams</td>
<td>TN</td>
</tr>
<tr>
<td>Joseph P. Gauthier</td>
<td>CA</td>
</tr>
</tbody>
</table>

(9 rows)
```
Table-Reference

Syntax

```
[[database.]schema.]table [AS alias]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>table</code></td>
<td>A table in the logical schema.</td>
</tr>
<tr>
<td><code>[AS] alias</code></td>
<td>A temporary name used for references to <code>table</code>.</td>
</tr>
</tbody>
</table>

Joined-Table

Specifies how to join tables.

Syntax

```
table-reference [join-type] JOIN table-reference[ TABLESAMPLE(sampling-pct) ] [ ON join-predicate ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table-reference</code></td>
<td>A table or another <code>joined-table</code>.</td>
</tr>
<tr>
<td><code>join-type</code></td>
<td>Valid Values:</td>
</tr>
<tr>
<td></td>
<td>• <code>INNER</code> (default). <code>INNER</code> JOIN is equivalent to a query that specifies its join predicate in a WHERE clause.</td>
</tr>
<tr>
<td></td>
<td>• <code>LEFT</code> [ <code>OUTER</code> ]</td>
</tr>
</tbody>
</table>
- **RIGHT [ OUTER ]
- **FULL [ OUTER ]
- **NATURAL
- **CROSS

**TABLESAMPLE**

Specifies to use simple random sampling to return an approximate percentage of records. All rows in the total potential return set are equally eligible to be included in the sampling. Vertica performs this sampling before other filters in the query are applied. The number of records returned is not guaranteed to be the exact percentage of records defined by `sampling-pct`.

The **TABLESAMPLE** option is valid only with user-defined tables and Data Collector (DC) tables. Views and system tables are not supported.

**sampling-pct**

Specifies the percentage of records to be returned as a part of sampling. The value must be greater than 0 and less than 100.

**ON join-predicate**

An equi-join based on one or more columns in the joined tables. invalid for NATURAL and CROSS joins, required for all other join types.

### Alternative JOIN Syntax Options

Vertica supports two older join syntax conventions:

- Table joins specified by join predicate in a WHERE clause
- Table joins specified by a USING clause

For details, see Join Syntax in Analyzing Data.

### Examples

The following SELECT statement qualifies its JOIN clause with the **TABLESAMPLE** option:

```sql
=> SELECT user_id.id, user_name.name FROM user_name TABLESAMPLE(50)
    JOIN user_id TABLESAMPLE(50) ON user_name.id = user_id.id;
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>-----------</td>
</tr>
</tbody>
</table>

---

---
GROUP BY Clause

Use the GROUP BY clause with aggregate functions in a SELECT statement to collect data across multiple records. Vertica groups the results into one or more sets of rows that match an expression.

The GROUP BY clause without aggregates is similar to using SELECT DISTINCT. **ROLLUP** is an extension to the GROUP BY clause. ROLLUP performs subtotal aggregations.

Syntax

```sql
GROUP BY [/*+GBYTYPE(algorithm)*/] expression-or-aggregate-expression [ , ... ]
```

Arguments

| /*+GBYTYPE(algorithm)*/ | Specifies which algorithm has precedence for implementing this GROUP BY clause, over the algorithm the Vertica query optimizer might otherwise choose. You can set **algorithm** to one of the following values:
| --- | --- |
| HASH: GROUPBY HASH algorithm
| PIPE: GROUPBY PIPELINED algorithm
For more information about both algorithms, see GROUP BY Implementation Options. |

| expression-or-aggregate-expression | An **expression** is:
| --- | --- |
| Any expression, including constants and column references in the tables specified in the FROM clause. For example:
| column1, ..., column_n, (expression) |
An **aggregate-expression** is:
- An ordered list of columns, expressions, CUBE, GROUPING SETS, or ROLLUP aggregates.

You can include CUBE and ROLLUP aggregates within a GROUPING SETS aggregate. CUBE and ROLLUP aggregates can result in a large amount of output. In that case, use GROUPING SETS to return only certain results.

You cannot include any aggregates within a CUBE or ROLLUP expression.

You can append multiple GROUPING SETS, CUBE, or ROLLUP aggregates in the same query. Examples:

```sql
GROUP BY a,b,c,d, ROLLUP(a,b)
GROUP BY a,b,c,d, CUBE((a,b),c,d)
GROUP BY a,b,c,d, CUBE(a,b), ROLLUP (c,d)
GROUP BY ROLLUP(a), CUBE(b), GROUPING SETS(c)
GROUP BY a,b,c,d, GROUPING SETS (((a,d),(b,c),CUBE(a,b)))
GROUP BY a,b,c,d, GROUPING SETS (((a,d),(b,c),(a,b),(a),(b),()))
```

Usage Considerations

- `expression` cannot include aggregate functions. However, you can use the GROUP BY clause with CUBE, GROUPING SETS, and ROLLUP to return summary values for each group.

- When you create a GROUP BY clause, you must include all non-aggregated columns that appear in the SELECT list.

- If the GROUP BY clause includes a WHERE clause, Vertica ignores all rows that do not satisfy the WHERE clause.

Examples

This example shows how to use the WHERE clause with GROUP BY. In this case, the example retrieves all employees whose last name begins with S, and ignores all rows that do not meet this criteria. The GROUP BY clause uses the ILIKE function to retrieve only last names beginning with S. The aggregate function SUM computes the total vacation days for each group.

```sql
=> SELECT employee_last_name, SUM(vacation_days)
    FROM employee_dimension
    WHERE employee_last_name ILIKE 'S%'```
The GROUP BY clause in the following example groups results by vendor region, and vendor region’s biggest deal:

```sql
=> SELECT vendor_region, MAX(deal_size) AS "Biggest Deal"
    FROM vendor_dimension
    GROUP BY vendor_region;
    vendor_region | Biggest Deal
-----------------|-------------
    East          | 990889
    Midwest       | 699163
    Northwest     | 76101
    South         | 854136
    Southwest     | 609807
    West          | 964005
(6 rows)
```

The following query modifies the previous one with a HAVING clause, which specifies to return only groups whose maximum deal size exceeds $900,000:

```sql
=> SELECT vendor_region, MAX(deal_size) AS "Biggest Deal"
    FROM vendor_dimension
    GROUP BY vendor_region
    HAVING MAX(deal_size) > 900000;
    vendor_region | Biggest Deal
-----------------|-------------
    East          | 990889
    West          | 964005
(2 rows)
```

The GROUP BY clause without aggregates is similar to using SELECT DISTINCT. For example, the following two queries return the same results:

```sql
=> SELECT DISTINCT household_id FROM customer_dimension;
=> SELECT household_id FROM customer_dimension GROUP BY household_id;
```

See Also

- CUBE Aggregate
- GROUP_ID
- GROUPING
ROLLUP Aggregate

Automatically performs subtotal aggregations as an extension to the GROUP BY clause. ROLLUP performs these aggregations across multiple dimensions, at different levels, within a single SQL query.

You can use the ROLLUP clause with three grouping functions:

- GROUPING
- GROUP_ID
- GROUPING_ID

Syntax

ROLLUP grouping-expression[, ...]

Parameters

<table>
<thead>
<tr>
<th>group-expression</th>
<th>One or both of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example:</td>
</tr>
<tr>
<td></td>
<td>column1, (column2+1), column3+column4</td>
</tr>
<tr>
<td></td>
<td>- A multilevel expression, one of the following:</td>
</tr>
<tr>
<td></td>
<td>▪ ROLLUP</td>
</tr>
<tr>
<td></td>
<td>▪ CUBE</td>
</tr>
<tr>
<td></td>
<td>▪ GROUPING SETS</td>
</tr>
</tbody>
</table>
Restrictions

GROUP BY ROLLUP does not sort results. To sort data, an ORDER BY clause must follow the GROUP BY clause.

Levels of Aggregation

If $n$ is the number of grouping columns, ROLLUP creates $n+1$ levels of subtotals and grand total. Because ROLLUP removes the right-most column at each step, specify column order carefully.

Suppose that ROLLUP(A, B, C) creates four groups:

- (A, B, C)
- (A, B)
- (A)
- ()

Because ROLLUP removes the right-most column at each step, there are no groups for (A, C) and (B, C).

If you enclose two or more columns in parentheses, GROUP BY treats them as a single entity. For example:

- ROLLUP(A, B, C) creates four groups:
  - (A, B, C)
  - (A, B)
  - (A)
  - ()

- ROLLUP((A, B), C) treats (A, B) as a single entity and creates three groups:
  - (A, B, C)
  - (A, B)
  - ()

Example: Aggregating the Full Data Set

The following example shows how to use the GROUP BY clause to determine family expenses for electricity and books over several years. The SUM aggregate function computes the total amount of money spent in each category per year.

Suppose you have a table that contains information about family expenses for books and electricity:
For the expenses table, ROLLUP computes the subtotals in each category between 2005–2007:

- Books: $99.96
- Electricity: $449.96
- Grand total: $549.92.

Use the ORDER BY clause to sort the results:

```
=> SELECT * FROM expenses ORDER BY Category, Year;
 Year | Category | Amount
--------+---------+-------
 2005  | Books   | 39.98 
 2007  | Books   | 29.99 
 2008  | Books   | 29.99 
 2005  | Electricity | 109.99
 2006  | Electricity | 109.99
 2007  | Electricity | 229.98
```

Example: Using ROLLUP with the HAVING Clause

This example shows how to use the HAVING clause with ROLLUP to restrict the GROUP BY results. The following query produces only those ROLLUP categories where year is subtotalled, based on the expression in the GROUPING function:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
 GROUP BY ROLLUP(Category, Year) ORDER BY 1,2, GROUPING_ID();
 Category | Year | SUM
--------+------+-------
 Books   | 2005 | 39.98 
 Books   | 2007 | 29.99 
 Books   | 2008 | 29.99 
 Books   |      | 99.96 
 Electricity | 2005 | 109.99
 Electricity | 2006 | 109.99
 Electricity | 2007 | 229.98
 Electricity |      | 449.96
 |      | 549.92
```
The next example rolls up on (Category, Year), but not on the full results. The GROUPING_ID function specifies to aggregate less than three levels:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
  GROUP BY ROLLUP(Category,Year) HAVING GROUPING_ID(Category,Year)<3
  ORDER BY 1, 2, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
</tr>
</tbody>
</table>

See Also

- Data Aggregation
- CUBE Aggregate
- GROUPING
- GROUP_ID
- GROUPING_ID
- GROUP BY Clause
- GROUPING SETS Aggregate

GROUP_ID

Uniquely identifies duplicate sets for GROUP BY queries that return duplicate grouping sets. This function returns one or more integers, starting with zero (0), as identifiers.

For the number of duplicates \( n \) for a particular grouping, \( \text{GROUP_ID} \) returns a range of sequential numbers, 0 to \( n-1 \). For the first each unique group it encounters, \( \text{GROUP_ID} \) returns the value 0. If \( \text{GROUP_ID} \) finds the same grouping again, the function returns 1, then returns 2 for the next found grouping, and so on.

**Note:** Use \( \text{GROUP_ID} \) only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.
Behavior Type

Immutable
Syntax

GROUP_ID ()
Examples

This example shows how GROUP_ID creates unique identifiers when a query produces duplicate groupings. For an expenses table, the following query groups the results by category of expense and year and rolls up the sum for those two columns. The results have duplicate groupings for category and NULL. The first grouping has a GROUP_ID of 0, and the second grouping has a GROUP_ID of 1.

```sql
=> SELECT Category, Year, SUM(Amount), GROUPING_ID(Category, Year), GROUP_ID() FROM expenses GROUP BY Category, ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING_ID</th>
<th>GROUP_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
See Also

- CUBE Aggregate
- GROUPING
- GROUPING_ID
- GROUPING SETS Aggregate
- GROUP BY Clause
- ROLLUP Aggregate

GROUPING

Disambiguates the use of NULL values when GROUP BY queries with multilevel aggregates generate NULL values to identify subtotals in grouping columns. Such NULL values from the original data can also occur in rows. GROUPING returns 1, if the value of expression is:

- NULL, representing an aggregated value
- 0 for any other value, including NULL values in rows

Note: Use GROUPING only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.
Behavior Type

Immutable
Syntax

GROUPING ( expression )
## Parameters

| $expression$ | An expression in the GROUP BY clause |
Examples

The following query uses the GROUPING function, taking one of the GROUP BY expressions as an argument. For each row, GROUPING returns one of the following:

- 0: The column is part of the group for that row
- 1: The column is not part of the group for that row

The 1 in the GROUPING(Year) column for electricity and books indicates that these values are subtotals. The right-most column values for both GROUPING(Category) and GROUPING(Year) are 1. This value indicates that neither column contributed to the GROUP BY. The final row represents the total sales.

```sql
=> SELECT Category, Year, SUM(Amount),
    GROUPING(Category), GROUPING(Year) FROM expenses
GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2008</td>
<td>449.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>549.92</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
See Also

- CUBE Aggregate
- GROUP_ID
- GROUPING_ID
- GROUPING SETS Aggregate
- GROUP BY Clause
- ROLLUP Aggregate

GROUPING_ID

Concatenates the set of Boolean values generated by the GROUPING function into a bit vector. GROUPING_ID treats the bit vector as a binary number and returns it as a base-10 value that identifies the grouping set combination.

By using GROUPING_ID you avoid the need for multiple, individual GROUPING functions. GROUPING_ID simplifies row-filtering conditions, because rows of interest are identified using a single return from GROUPING_ID = n. Use GROUPING_ID to identify grouping combinations.

Note: Use GROUPING_ID only in SELECT statements that contain a GROUP BY aggregate: CUBE, GROUPING SETS, and ROLLUP.
Behavior Type

Immutable
### Syntax

GROUPING_ID ( [expression[,...] ] )

<table>
<thead>
<tr>
<th>expression</th>
<th>An expression that matches one of the expressions in the GROUP BY clause.</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>If the GROUP BY clause includes a list of expressions, GROUPING_ID returns a number corresponding to the GROUPING bit vector associated with a row.</td>
</tr>
</tbody>
</table>
Examples

This example shows how calling GROUPING_ID without an expression returns the GROUPING bit vector associated with a full set of multilevel aggregate expressions. The GROUPING_ID value is comparable to GROUPING_ID(a, b) because GROUPING_ID() includes all columns in the GROUP BY ROLLUP:

```sql
=> SELECT a, b, COUNT(*), GROUPING_ID() FROM T GROUP BY ROLLUP(a, b);
```

In the following query, the GROUPING(Category) and GROUPING(Year) columns have three combinations:

- 0,0
- 0,1
- 1,1

```sql
=> SELECT Category, Year, SUM(Amount),
   GROUPING(Category), GROUPING(Year) FROM expenses
   GROUP BY ROLLUP(Category, Year) ORDER BY Category, Year, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>549.92</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

GROUPING_ID converts these values as follows:

<table>
<thead>
<tr>
<th>Binary Set Values</th>
<th>Decimal Equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>Category, Year</td>
</tr>
</tbody>
</table>

The following query returns the single number for each GROUP BY level that appears in the grid column:

```sql
SELECT Category, Year, SUM(Amount),
   GROUPING(Category), GROUPING(Year)
FROM expenses
GROUP BY ROLLUP(Category, Year)
ORDER BY Category, Year, GROUPING_ID();
```
=> SELECT Category, Year, SUM(Amount),
    GROUPING(Category), GROUPING(Year), GROUPING_ID(Category, Year) AS gr_id
FROM expenses GROUP BY ROLLUP(Category, Year);

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUPING</th>
<th>GROUPING</th>
<th>gr_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>549.92</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

The gr_id value determines the GROUP BY level for each row:

<table>
<thead>
<tr>
<th>GROUP BY Level</th>
<th>GROUP BY Row Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Total sum</td>
</tr>
<tr>
<td>1</td>
<td>Category</td>
</tr>
<tr>
<td>0</td>
<td>Category, year</td>
</tr>
</tbody>
</table>

You can also use the DECODE function to give the values more meaning by comparing each search value individually:

=> SELECT Category, Year, SUM(AMOUNT), DECODE(GROUPING_ID(Category, Year),
    3, 'Total',
    1, 'Category',
    0, 'Category,Year')
AS GROUP_NAME FROM expenses GROUP BY ROLLUP(Category, Year);

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
<th>GROUP_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
<td>Category</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
<td>Category</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
<td>Total</td>
</tr>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
<td>Category,Year</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
<td>Category,Year</td>
</tr>
</tbody>
</table>
See Also

- CUBE Aggregate
- GROUP_ID
- GROUPING
- GROUPING SETS Aggregate
- GROUP BY Clause
- ROLLUP Aggregate

CUBE Aggregate

Automatically performs all possible aggregations of the specified columns, as an extension to the GROUP BY clause.

You can use the ROLLUP clause with three grouping functions:

- GROUPING
- GROUP_ID
- GROUPING_ID

Syntax

GROUP BY `group-expression[, ...]`

Parameters

<table>
<thead>
<tr>
<th><code>group-expression</code></th>
<th>One or both of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example:</td>
</tr>
<tr>
<td></td>
<td>column1, (column2+1), column3+column4</td>
</tr>
</tbody>
</table>
A multilevel expression, one of the following:
- ROLLUP
- CUBE
- GROUPING SETS

**Restrictions**

- GROUP BY CUBE does not order data. If you want to sort data, use the ORDER BY Clause. The ORDER BY clause must come after the GROUP BY clause.

- You can use CUBE inside a GROUPING SETS expression, but not inside a ROLLUP expression or another CUBE expression.

**Levels of CUBE Aggregation**

If \( n \) is the number of grouping columns, CUBE creates \( 2^n \) levels of aggregations. For example: CUBE \( (A, B, C) \) creates all possible permutations, resulting in eight groups:

- \( (A, B, C) \)
- \( (A, B) \)
- \( (A, C) \)
- \( (B, C) \)
- \( (A) \)
- \( (B) \)
- \( (C) \)
- \( () \)

If you increase the number of CUBE columns, the number of CUBE groupings increases exponentially. The CUBE query may be resource intensive and produce combinations that are not of interest. In that case, consider using the GROUPING SETS Aggregate, which allows you to choose specific groupings.
Example: Using CUBE to Return All Groupings

Suppose you have a table that contains information about family expenses for books and electricity:

```sql
=> SELECT * FROM expenses ORDER BY Category, Year;

<table>
<thead>
<tr>
<th>Year</th>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Books</td>
<td>39.98</td>
</tr>
<tr>
<td>2007</td>
<td>Books</td>
<td>29.99</td>
</tr>
<tr>
<td>2008</td>
<td>Books</td>
<td>29.99</td>
</tr>
<tr>
<td>2005</td>
<td>Electricity</td>
<td>109.99</td>
</tr>
<tr>
<td>2006</td>
<td>Electricity</td>
<td>109.99</td>
</tr>
<tr>
<td>2007</td>
<td>Electricity</td>
<td>229.98</td>
</tr>
</tbody>
</table>
```

To aggregate the data by both Category and Year using the CUBE aggregate:

```sql
=> SELECT Category, Year, SUM(Amount) FROM expenses
   GROUP BY CUBE(Category, Year) ORDER BY 1, 2, GROUPING_ID();

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electricity</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electricity</td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td>Electricity</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
</tr>
<tr>
<td></td>
<td>2005</td>
<td>149.97</td>
</tr>
<tr>
<td></td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td></td>
<td>2007</td>
<td>259.97</td>
</tr>
<tr>
<td></td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
</tr>
</tbody>
</table>
```

The results include subtotals for each category and year, and a grand total ($549.92).

Example: Using CUBE with the HAVING Clause

This example shows how you can restrict the GROUP BY results, use the HAVING clause with the CUBE aggregate. This query returns only the category totals and the full total:

```sql
=> SELECT Category, Year, SUM(Amount) FROM expenses
   GROUP BY CUBE(Category, Year) HAVING GROUPING(Year)=1;

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electricity</td>
<td></td>
<td>449.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>549.92</td>
</tr>
</tbody>
</table>
```

The next query returns only the aggregations for the two categories for each year. The GROUPING ID function specifies to omit the grand total ($549.92):
```sql
SELECT Category, Year, SUM(Amount) FROM expenses
GROUP BY CUBE(Category,Year) HAVING GROUPING_ID(Category,Year)<2
ORDER BY 1, 2, GROUPING_ID();
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Year</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Books</td>
<td>2005</td>
<td>39.98</td>
</tr>
<tr>
<td>Books</td>
<td>2007</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td>2008</td>
<td>29.99</td>
</tr>
<tr>
<td>Books</td>
<td></td>
<td>99.96</td>
</tr>
<tr>
<td>Electrical</td>
<td>2005</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2006</td>
<td>109.99</td>
</tr>
<tr>
<td>Electrical</td>
<td>2007</td>
<td>229.98</td>
</tr>
<tr>
<td>Electrical</td>
<td></td>
<td>449.96</td>
</tr>
</tbody>
</table>

See Also

- Data Aggregation
- GROUP BY Clause
- GROUP_ID
- GROUPING
- GROUPING_ID
- GROUPING SETS Aggregate
- ROLLUP Aggregate

**GROUPING SETS Aggregate**

The GROUPING SETS aggregate is an extension to the GROUP BY clause that automatically performs subtotal aggregations on groupings that you specify.

You can use the GROUPING SETS clause with three grouping functions:

- GROUPING
- GROUP_ID
- GROUPING_ID

To sort data, use the ORDER BY clause. The ORDER BY clause must follow the GROUP BY clause.
Syntax

GROUP BY group-expression[,...]

Parameters

<table>
<thead>
<tr>
<th>group-expression</th>
<th>One or both of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• An expression that is not an aggregate or a grouping function that includes constants and column references in FROM-specified tables. For example:</td>
</tr>
<tr>
<td></td>
<td>column1, (column2+1), column3+column4</td>
</tr>
<tr>
<td></td>
<td>• A multilevel expression, one of the following:</td>
</tr>
<tr>
<td></td>
<td>▪ ROLLUP</td>
</tr>
<tr>
<td></td>
<td>▪ CUBE</td>
</tr>
<tr>
<td></td>
<td>▪ GROUPING SETS</td>
</tr>
</tbody>
</table>

Defining the Groupings

GROUPING SETS allows you to specify exactly which groupings you want in the results. You can also concatenate the groupings as follows:

The following example clauses result in the groupings shown.

<table>
<thead>
<tr>
<th>This clause...</th>
<th>Defines groupings...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...GROUP BY GROUPING SETS(A,B,C,D)...</td>
<td>(A), (B), (C), (D)</td>
</tr>
<tr>
<td>...GROUP BY GROUPING SETS((A),(B),(C),(D))...</td>
<td>(A), (B), (C), (D)</td>
</tr>
<tr>
<td>...GROUP BY GROUPING SETS((A,B,C,D))...</td>
<td>(A, B, C, D)</td>
</tr>
<tr>
<td>...GROUP BY GROUPING SETS(A,B),GROUPING SETS(C,D)...</td>
<td>(A, C), (B, C), (A, D), (B, C)</td>
</tr>
<tr>
<td>...GROUP BY GROUPING SETS((A,B)),GROUPING SETS(C,D)...</td>
<td>(A, B, C), (A, B, D)</td>
</tr>
<tr>
<td>...GROUP BY GROUPING SETS(A,B),GROUPING SETS(ROLLUP(C,D))...</td>
<td>(A,B), (A,B,C), (A,B,C,D)</td>
</tr>
</tbody>
</table>
This clause...

...GROUP BY A, B, C, GROUPING SETS(ROLLUP(C, D))...

Defines groupings...

(A, B, C, D), (A, B, C), (A, B, C)

The clause contains two groups (A, B, C). In the HAVING clause, use the GROUP_ID function as a predicate, to eliminate the second grouping.

Example: Selecting Groupings

This example shows how to select only those groupings you want. Suppose you want to aggregate on columns only, and you do not need the grand total. The first query omits the total. In the second query, you add () to the GROUPING SETS list to get the total. Use the ORDER BY clause to sort the results by grouping:

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
    GROUP BY GROUPING SETS((Category, Year), (Year))
    ORDER BY 1, 2, GROUPING_ID();

Category | Year | SUM
---------|------|-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
      | 2005 | 149.97
      | 2006 | 109.99
      | 2007 | 259.97
      | 2008 | 29.99
```

```
=> SELECT Category, Year, SUM(Amount) FROM expenses
    GROUP BY GROUPING SETS((Category, Year), (Year), ())
    ORDER BY 1, 2, GROUPING_ID();

Category | Year | SUM
---------|------|-----
Books    | 2005 | 39.98
Books    | 2007 | 29.99
Books    | 2008 | 29.99
Electrical | 2005 | 109.99
Electrical | 2006 | 109.99
Electrical | 2007 | 229.98
      | 2005 | 149.97
      | 2006 | 109.99
      | 2007 | 259.97
      | 2008 | 29.99
      | 549.92
```
See Also

- Data Aggregation
- CUBE Aggregate
- GROUPING
- GROUP_ID
- GROUPING_ID
- GROUP BY Clause
- ROLLUP Aggregate

HAVING Clause

Restricts the results of a GROUP BY Clause.

Syntax

HAVING condition [, ...]

Parameters

| condition | Must unambiguously reference a grouping column, unless the reference appears within an aggregate function |

Notes

- Semantically, the HAVING clause occurs after the GROUP BY operation.
- You can use expressions in the HAVING clause.
- The HAVING clause was added to the SQL standard because you cannot use WHERE with Aggregate Functions.
Example

The following example returns the employees with salaries greater than $50,000:

```sql
=> SELECT employee_last_name, MAX(annual_salary) as "highest_salary"
   FROM employee_dimension
   GROUP BY employee_last_name
   HAVING MAX(annual_salary) > 50000;
```

<table>
<thead>
<tr>
<th>employee_last_name</th>
<th>highest_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bauer</td>
<td>920149</td>
</tr>
<tr>
<td>Brown</td>
<td>569079</td>
</tr>
<tr>
<td>Campbell</td>
<td>649998</td>
</tr>
<tr>
<td>Carcetti</td>
<td>195175</td>
</tr>
<tr>
<td>Dobisz</td>
<td>840902</td>
</tr>
<tr>
<td>Farmer</td>
<td>804890</td>
</tr>
<tr>
<td>Fortin</td>
<td>481490</td>
</tr>
<tr>
<td>García</td>
<td>811231</td>
</tr>
<tr>
<td>Garnett</td>
<td>963104</td>
</tr>
<tr>
<td>Gauthier</td>
<td>927335</td>
</tr>
</tbody>
</table>

(10 rows)

INTERSECT Clause

Calculates the intersection of the results of two or more SELECT queries. INTERSECT returns distinct values by both the query on the left and right sides of the INTERSECT operand.

Syntax

```
select ... INTERSECT select ...
[ INTERSECT select ]...
[ ORDER BY { column-name | ordinal-number } [ ASC | DESC ] ]
[ , ... ]
[ LIMIT { integer | ALL } ]
[ OFFSET integer ]
```

Notes

- Use the INTERSECT clause to return all elements that are common to the results of all the SELECT queries. The INTERSECT query operates on the results of two or more SELECT queries. INTERSECT returns only the rows that are returned by all the specified queries.

- You cannot use the ALL keyword with an INTERSECT query.
The results of each SELECT query must be union compatible; they must return the same number of columns, and the corresponding columns must have compatible data types. For example, you cannot use the INTERSECT clause on a column of type INTEGER and a column of type VARCHAR. If the SELECT queries do not meet these criteria, Vertica returns an error.

**Note:** The Data Type Coercion Chart lists the data types that can be cast to other data types. If one data type can be cast to the other, those two data types are compatible.

Order the results of an INTERSECT operation by using an ORDER BY clause. In the ORDER BY list, specify the column names from the leftmost SELECT statement or specify integers that indicate the position of the columns by which to sort.

You can use INTERSECT in FROM, WHERE, and HAVING clauses.

The rightmost ORDER BY, LIMIT, or OFFSET clauses in an INTERSECT query do not need to be enclosed in parentheses because the rightmost query specifies that Vertica perform the operation on the results of the INTERSECT operation. Any ORDER BY, LIMIT, or OFFSET clauses contained in SELECT queries that appear earlier in the INTERSECT query must be enclosed in parentheses.

The order by column names is from the first select.

Vertica supports INTERSECT noncorrelated subquery predicates. For example:

```sql
=> SELECT * FROM T1
   WHERE T1.x IN
       (SELECT MAX(c1) FROM T2
        INTERSECT
         SELECT MAX(cc1) FROM T3
        INTERSECT
         SELECT MAX(d1) FROM T4);
```

### Examples

Consider the following three tables:

**Company_A**

<table>
<thead>
<tr>
<th>id</th>
<th>emp_name</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>auto parts</td>
<td>1000</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
<td>auto parts</td>
<td>2500</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>floral</td>
<td>500</td>
</tr>
<tr>
<td>3214</td>
<td>Smithson</td>
<td>sporting goods</td>
<td>1500</td>
</tr>
</tbody>
</table>

**Company_B**
The following query returns the IDs and last names of employees that exist in both Company_A and Company_C:

```sql
=> SELECT id, emp_lname FROM Company_A
    INTERSECT
    SELECT id, emp_lname FROM Company_C;
```
```
id | emp_lname
----------
3214 | Smithson
9012 | Katherine
```
(2 rows)

The following query returns the same two employees in descending order of sales:

```sql
=> SELECT id, emp_lname, sales FROM Company_A
    INTERSECT
    SELECT id, emp_lname, sales FROM Company_B
    ORDER BY sales DESC;
```
```
id | emp_lname | sales
----------
3214 | Smithson | 1500
9012 | Katherine | 500
```
(2 rows)

You can also use the integer that represents the position of the sales column (3) to return the same result:

```sql
=> SELECT id, emp_lname, sales FROM Company_A
    INTERSECT
    SELECT id, emp_lname, sales FROM Company_B
    ORDER BY 3 DESC;
```
```
id | emp_lname | sales
----------
3214 | Smithson | 1500
9012 | Katherine | 500
```
(2 rows)
The following query returns the employee who works for both companies whose sales in Company_B are greater than 1000:

```sql
=> SELECT id, emp_lname, sales FROM Company_A
    INTERSECT
    (SELECT id, emp_lname, sales FROM company_B WHERE sales > 1000)
    ORDER BY sales DESC;
id | emp_lname | sales
--------+------------+-----
3214 | Smithson  | 1500
(1 row)
```

In the following query returns the ID and last name of the employee who works for all three companies:

```sql
=> SELECT id, emp_lname FROM Company_A
    INTERSECT
    SELECT id, emp_lname FROM Company_B
    INTERSECT
    SELECT id, emp_lname FROM Company_C;
id | emp_lname
--------+------------
3214 | Smithson
(1 row)
```

The following query shows the results of a mismatched data types; these two queries are not union compatible:

```sql
=> SELECT id, emp_lname FROM Company_A
    INTERSECT
    SELECT emp_lname, id FROM Company_B;
ERROR 3429: For 'INTERSECT', types int and varchar are inconsistent
DETAIL: Columns: id and emp_lname
```

Using the VMart example database, the following query returns information about all Connecticut-based customers who bought items online and whose purchase amounts were between $400 and $500:

```sql
=> SELECT customer_key, customer_name from public.customer_dimension
    WHERE customer_key IN (SELECT customer_key
                       FROM online_sales.online_sales_fact
                       WHERE sales_dollar_amount > 400
                    INTERSECT
                    SELECT customer_key FROM online_sales.online_sales_fact
                       WHERE sales_dollar_amount < 500)
                    AND customer_state = 'CT';
customer_key | customer_name
--------------+------------------
39           | Sarah S. Winkler
44           | Meghan H. Overstreet
70           | Jack X. Cleveland
103          | Alexandra I. Vu
110          | Matt. Farmer
173          | Mary R. Reyes
```
The previous query returns the same data as:

```sql
=> SELECT customer_key, customer_name FROM public.customer_dimension
    WHERE customer_key IN (SELECT customer_key
                            FROM online_sales.online_sales_fact
                            WHERE sales_dollar_amount > 400
                            AND sales_dollar_amount < 500)
    AND customer_state = 'CT';
```

See Also

- SELECT
- EXCEPT Clause
- UNION Clause
- Subqueries

INTO TABLE Clause

Creates a table from a query result set.

Syntax

**Permanent table**

INTO [TABLE] [[database.].schema.]table

**Temporary table**

INTO [scope] TEMP[ORARY] [TABLE] [[database.].schema.]table
...
[ ON COMMIT { PRESERVE | DELETE } ROWS ]
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>scope</strong></td>
<td>Specifies visibility of a temporary table definition:</td>
</tr>
<tr>
<td></td>
<td>- GLOBAL (default): The table definition is visible to all sessions, and persists until you explicitly drop the table.</td>
</tr>
<tr>
<td></td>
<td>- LOCAL: The table definition is visible only to the session in which it is created, and is dropped when the session ends.</td>
</tr>
<tr>
<td></td>
<td>Regardless of this setting, retention of temporary table data is set by the keywords ON COMMIT DELETE ROWS and ON COMMIT PRESERVE ROWS (see below).</td>
</tr>
<tr>
<td></td>
<td>For more information, see <a href="#">Creating Temporary Tables</a> in the Administrator's Guide.</td>
</tr>
<tr>
<td><strong>[database.]schema</strong></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example:</td>
</tr>
<tr>
<td></td>
<td><code>myschema.thisDbObject</code></td>
</tr>
<tr>
<td></td>
<td>If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><strong>table</strong></td>
<td>The name of the table to create.</td>
</tr>
<tr>
<td>**ON COMMIT { PRESERVE</td>
<td>DELETE } ROWS**</td>
</tr>
<tr>
<td></td>
<td>- DELETE (default) marks the temporary table for transaction-scoped data. Vertica removes all table data after each commit.</td>
</tr>
<tr>
<td></td>
<td>- PRESERVE marks the temporary table for session-scoped data, which is preserved beyond the lifetime of a single transaction. Vertica removes all table data when the session ends.</td>
</tr>
</tbody>
</table>
Examples

The following SELECT statement has an INTO TABLE clause that creates table newTable from customer_dimension:

```sql
=> SELECT * INTO TABLE newTable FROM customer_dimension;
```

The following SELECT statement creates temporary table newTempTable. By default, temporary tables are created at a global scope, so its definition is visible to other sessions and persists until it is explicitly dropped. No customer_dimension data is copied into the new table, and Vertica issues a warning accordingly:

```sql
=> SELECT * INTO TEMP TABLE newTempTable FROM customer_dimension;
WARNING 4102: No rows are inserted into table "public"."newTempTable" because ON COMMIT DELETE ROWS is the default for create temporary table
HINT: Use "ON COMMIT PRESERVE ROWS" to preserve the data in temporary table
CREATE TABLE
```

The following SELECT statement creates local temporary table newTempTableLocal. This table is visible only to the session in which it was created, and is automatically dropped when the session ends. The INTO TABLE clause includes ON COMMIT PRESERVE ROWS, so Vertica copies all selection data into the new table:

```sql
=> SELECT * INTO LOCAL TEMP TABLE newTempTableLocal ON COMMIT PRESERVE ROWS
FROM customer_dimension;
CREATE TABLE
```

LIMIT Clause

Specifies the maximum number of result set rows to return.

Syntax

```
LIMIT { rows | ALL }
```

Parameters

<table>
<thead>
<tr>
<th>rows</th>
<th>The maximum number of rows to return.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>(default) Returns all rows.</td>
</tr>
</tbody>
</table>
Dependencies

- Use an ORDER BY clause with LIMIT. Otherwise, the query returns an undefined subset of the result set. For example, the following SELECT statement omits ORDER BY. Successive iterations of this query are liable to return a different set of five records from the customer_dimension table:

```
=> SELECT customer_name, customer_city FROM customer_dimension LIMIT 5;
   customer_name | customer_city
---------------|---------------
Craig S. Robinson | Fayetteville
Mark M. Kramer | Joliet
Barbara S. Farmer | Alexandria
Julie S. McNulty | Grand Prairie
Meghan R. Garcia | Athens
(5 rows)
```

In contrast, the following SELECT statement includes an ORDER BY clause and returns a consistent set of results:

```
=> SELECT customer_name, customer_city FROM customer_dimension
   ORDER BY customer_city, customer_name LIMIT 5;
   customer_name | customer_city
---------------|---------------
Alexander . Dobisz | Abilene
Alexander B. McCabe | Abilene
Alexander J. Goldberg | Abilene
Alexander M. Fortin | Abilene
Alexander P. Moore | Abilene
(5 rows)
```

- LIMIT must follow the SELECT statement's ORDER BY clause.
- When a SELECT statement specifies both LIMIT and OFFSET, Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

MATCH Clause

A SQL extension that lets you screen large amounts of historical data in search of event patterns, the MATCH clause provides subclasses for analytic partitioning and ordering and matches rows from the result table based on a pattern you define.

You specify a pattern as a regular expression, which is composed of event types defined in the DEFINE subclause, where each event corresponds to a row in the input table. Then you can search for the pattern within a sequence of input events. Pattern matching returns the contiguous sequence of rows that conforms to PATTERN subclause. For example, pattern P (A
B* C) consist of three event types: A, B, and C. When Vertica finds a match in the input table, the associated pattern instance must be an event of type A followed by 0 or more events of type B, and an event of type C.

Pattern matching is particularly useful for clickstream analysis where you might want to identify users' actions based on their Web browsing behavior (page clicks). A typical online clickstream funnel is:

Company home page -> product home page -> search -> results -> purchase online

Using the above clickstream funnel, you can search for a match on the user's sequence of web clicks and identify that the user:

- Landed on the company home page
- Navigated to the product page
- Ran a search
- Clicked a link from the search results
- Made a purchase

For examples that use this clickstream model, see Event Series Pattern Matching in Analyzing Data.

**Syntax**

```
MATCH ( [ PARTITION BY table_column ] ORDER BY table_column
... DEFINE event_name AS boolean_expr [, ...]
... PATTERN pattern_name AS ( regexp )
... [ ROWS MATCH { ALL EVENTS | FIRST EVENT } ] )
```

**Parameters**

| PARTITION BY | [Optional] Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. The partition clause partitions the data by matched patterns defined in the PATTERN subclause. For each partition, data is sorted by the ORDER BY clause. If the partition clause is omitted, the entire data set is considered a single partition. |
| ORDER BY     | Defines the window data scope in which the pattern, defined in the PATTERN subclause, is matched. For each partition, the order clause |
specifies how the input data is ordered for pattern matching.

**Note:** The `ORDER BY` clause is mandatory.

<table>
<thead>
<tr>
<th>DEFINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines the <strong>boolean</strong> expressions that make up the event types in the regular expressions. For example:</td>
</tr>
<tr>
<td>The <code>DEFINE</code> subclause accepts a maximum of 52 events. See <a href="#">Event Series Pattern Matching</a> in Machine Learning for Predictive Analytics for examples.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>event_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the name of the event to evaluate for each row; for example, Entry, Onsite, Purchase.</td>
</tr>
<tr>
<td><strong>Note:</strong> Event names are case insensitive and follow the same naming conventions as those used for tables and columns.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean_expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is an expression that returns true or false. <code>boolean_expr</code> can include <a href="#">Boolean Operators</a> and relational (comparison) operators. For example:</td>
</tr>
<tr>
<td>Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PATTERN pattern_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the name of the pattern, which you define in the PATTERN subclause; for example, <code>P</code> is the pattern name defined below:</td>
</tr>
<tr>
<td><strong>PATTERN P AS (...)</strong></td>
</tr>
<tr>
<td>A <code>PATTERN</code> is a <strong>search pattern</strong> that is comprised of a name and a regular expression.</td>
</tr>
<tr>
<td><strong>Note:</strong> Vertica supports one pattern per query.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>regexp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is a regular expression comprised of event types (defined in the <code>DEFINE</code> subclause), and one or more of the quantifiers below. When Vertica evaluates the <code>MATCH</code> clause, the regular expression identifies the rows that meet the expression criteria.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*?</td>
</tr>
<tr>
<td>+</td>
</tr>
<tr>
<td>+?</td>
</tr>
<tr>
<td>?</td>
</tr>
<tr>
<td>??</td>
</tr>
<tr>
<td>*+</td>
</tr>
<tr>
<td>++</td>
</tr>
<tr>
<td>?+</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**ROWS MATCH**

[Optional] Defines how to resolve more than one event evaluating to true for a single row.

- If you use ROWS MATCH ALL EVENTS, Vertica returns the following run-time error if more than one event evaluates to true for a single row:

  ```
  ERROR: pattern events must be mutually exclusive
  HINT: try using ROWS MATCH FIRST EVENT
  ```

- For ROWS MATCH FIRST EVENT, if more than one event evaluates to true for a single row, Vertica chooses the event defined first in the SQL statement to be the event it uses for the row.

**Pattern Semantic Evaluation**

- The semantic evaluating ordering of the SQL clauses is: FROM -> WHERE -> PATTERN MATCH -> SELECT.

- Data is partitioned as specified in the PARTITION BY clause. If the partition clause is omitted, the entire data set is considered a single partition.
For each partition, the order clause specifies how the input data is ordered for pattern matching.

Events are evaluated for each row. A row could have 0, 1, or N events evaluate to true. If more than one event evaluates to true for the same row, Vertica returns a run-time error unless you specify ROWS MATCH FIRST EVENT. If you specify ROWS MATCH FIRST EVENT and more than one event evaluates to TRUE for a single row, Vertica chooses the event that was defined first in the SQL statement to be the event it uses for the row.

Vertica performs pattern matching by finding the contiguous sequence of rows that conforms to the pattern defined in the PATTERN subclause. For each match, Vertica outputs the rows that contribute to the match. Rows not part of the match (do not satisfy one or more predicates) are not output.

Vertica reports only non-overlapping matches. If an overlap occurs, Vertica chooses the first match found in the input stream. After finding the match, Vertica looks for the next match, starting at the end of the previous match.

Vertica reports the longest possible match, not a subset of a match. For example, consider pattern: A*B with input: AAAB. Because A uses the greedy regular expression quantifier (*), Vertica reports all A inputs (AAAB), not AAB, AB, or B.

Notes and Restrictions

- DISTINCT and GROUP BY/HAVING clauses are not allowed in pattern match queries.

- The following expressions are not allowed in the DEFINE subclause:
  - Subqueries, such as DEFINE X AS c IN (SELECT c FROM table1)
  - Analytic functions, such as DEFINE X AS c < LEAD(1) OVER (ORDER BY 1)
  - Aggregate functions, such as DEFINE X AS c < MAX(1)

- You cannot use the same pattern name to define a different event; for example, the following is not allowed for X:

```sql
DEFINE X AS c1 < 3
X AS c1 >= 3
```
Used with MATCH clause, Vertica Pattern Matching Functions provide additional data about the patterns it finds. For example, you can use the functions to return values representing the name of the event that matched the input row, the sequential number of the match, or a partition-wide unique identifier for the instance of the pattern that matched.

Examples

For examples, see Event Series Pattern Matching in Machine Learning for Predictive Analytics.

See Also

- Pattern Matching Functions
- EVENT_NAME
- MATCH_ID
- PATTERN_ID

MINUS Clause

MINUS is an alias for EXCEPT.

OFFSET Clause

Omits a specified number of rows from the beginning of the result set.

Syntax

OFFSET rows

Parameters

| rows  | Specifies the number of result set rows to omit. |
Dependencies

- Use an ORDER BY clause with OFFSET. Otherwise, the query returns an undefined subset of the result set.

- OFFSET must follow the ORDER BY clause in a SELECT statement or UNION clause.

- When a SELECT statement or UNION clause specifies both LIMIT and OFFSET, Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

Example

The following query returns 14 rows from the customer_dimension table:

```sql
=> SELECT customer_name, customer_gender FROM customer_dimension
WHERE occupation='Dancer' AND customer_city = 'San Francisco' ORDER BY customer_name;
```

<table>
<thead>
<tr>
<th>customer_name</th>
<th>customer_gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy X. Lang</td>
<td>Female</td>
</tr>
<tr>
<td>Anna H. Li</td>
<td>Female</td>
</tr>
<tr>
<td>Brian O. Weaver</td>
<td>Male</td>
</tr>
<tr>
<td>Craig O. Pavlov</td>
<td>Male</td>
</tr>
<tr>
<td>Doug Z. Goldberg</td>
<td>Male</td>
</tr>
<tr>
<td>Harold S. Jones</td>
<td>Male</td>
</tr>
<tr>
<td>Jack E. Perkins</td>
<td>Male</td>
</tr>
<tr>
<td>Joseph W. Overstreet</td>
<td>Male</td>
</tr>
<tr>
<td>Kevin . Campbell</td>
<td>Male</td>
</tr>
<tr>
<td>Raja Y. Wilson</td>
<td>Male</td>
</tr>
<tr>
<td>Samantha O. Brown</td>
<td>Female</td>
</tr>
<tr>
<td>Steve H. Gauthier</td>
<td>Male</td>
</tr>
<tr>
<td>William . Nielson</td>
<td>Male</td>
</tr>
<tr>
<td>William Z. Roy</td>
<td>Male</td>
</tr>
</tbody>
</table>

(14 rows)

If you modify the previous query to specify an offset of 8 (OFFSET 8) clause, Vertica skips the first eight rows of the previous result set. The query returns the following results:

```sql
=> SELECT customer_name, customer_gender FROM customer_dimension
WHERE occupation='Dancer' AND customer_city = 'San Francisco' ORDER BY customer_name OFFSET 8;
```

<table>
<thead>
<tr>
<th>customer_name</th>
<th>customer_gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kevin . Campbell</td>
<td>Male</td>
</tr>
<tr>
<td>Raja Y. Wilson</td>
<td>Male</td>
</tr>
<tr>
<td>Samantha O. Brown</td>
<td>Female</td>
</tr>
<tr>
<td>Steve H. Gauthier</td>
<td>Male</td>
</tr>
<tr>
<td>William . Nielson</td>
<td>Male</td>
</tr>
<tr>
<td>William Z. Roy</td>
<td>Male</td>
</tr>
</tbody>
</table>

(6 rows)
ORDER BY Clause

Sorts a query result set on one or more columns.

Syntax

ORDER BY expression [ ASC | DESC ] [, ...]

Parameters

<table>
<thead>
<tr>
<th>expression</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Name or ordinal number of a SELECT list item. You cannot use an integer value for an ORDER BY clause that is inside an analytic function's OVER clause.</td>
</tr>
<tr>
<td></td>
<td>- Arbitrary expression formed from columns that do not appear in the SELECT list</td>
</tr>
<tr>
<td></td>
<td>- CASE expression</td>
</tr>
</tbody>
</table>

Notes

- The ordinal number refers to the position of the result column, counting from the left beginning at one. This makes it possible to order by a column that does not have a unique name. (You can assign a name to a result column using the AS clause.)

- While the user's current locale and collation sequence are used to compare strings and determine the results of the ORDER BY clause of a query, Vertica projection data is always stored sorted by the ASCII (binary) collating sequence.

- For INTEGER, INT, and DATE/TIME data types, NULL appears first (smallest) in ascending order.

- For FLOAT, BOOLEAN, CHAR, and VARCHAR, NULL appears last (largest) in ascending order.

- The ORDER BY clause may contain only columns or expressions that are in the window partition clause (see Window Partition Clause).
Examples

The follow example returns all the city and deal size for customer Metamedia, sorted by deal size in descending order.

```sql
=> SELECT customer_city, deal_size FROM customer_dimension WHERE customer_name = 'Metamedia'
   ORDER BY deal_size DESC;
```

<table>
<thead>
<tr>
<th>customer_city</th>
<th>deal_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>El Monte</td>
<td>4479561</td>
</tr>
<tr>
<td>Athens</td>
<td>3815416</td>
</tr>
<tr>
<td>Ventura</td>
<td>3792937</td>
</tr>
<tr>
<td>Peoria</td>
<td>3227765</td>
</tr>
<tr>
<td>Arvada</td>
<td>2671849</td>
</tr>
<tr>
<td>Coral Springs</td>
<td>2643674</td>
</tr>
<tr>
<td>Fontana</td>
<td>2374465</td>
</tr>
<tr>
<td>Rancho Cucamonga</td>
<td>2214002</td>
</tr>
<tr>
<td>Wichita Falls</td>
<td>2117962</td>
</tr>
<tr>
<td>Beaumont</td>
<td>1898295</td>
</tr>
<tr>
<td>Arvada</td>
<td>1321897</td>
</tr>
<tr>
<td>Waco</td>
<td>1026854</td>
</tr>
<tr>
<td>Joliet</td>
<td>945404</td>
</tr>
<tr>
<td>Hartford</td>
<td>445795</td>
</tr>
</tbody>
</table>

(14 rows)

The following example uses a transform function. It returns an error because the ORDER BY column is not in the window partition.

```sql
=> CREATE TABLE t(geom geometry(200), geog geography(200));
=> SELECT PolygonPoint(geom) OVER(PARTITION BY geom) AS SEL_0 FROM t ORDER BY geog;
ERROR 2521: Cannot specify anything other than user defined transforms and partitioning expressions in the ORDER BY list
```

The following example, using the same table, corrects this error.

```sql
=> SELECT PolygonPoint(geom) OVER(PARTITION BY geom) AS SEL_0 FROM t ORDER BY geom;
```

TIMESERIES Clause

Provides gap-filling and interpolation (GFI) computation, an important component of time series analytics computation. See Time Series Analytics in Analyzing Data for details and examples.
Syntax

```
TIMESERIES slice-time AS 'length-and-time-unit-exp' OVER (
... [ PARTITION BY (column-expr[,]) ]
... ORDER BY time-expr )
... [ ORDER BY table-column[, ] ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>slice-time</code></td>
<td>A time column produced by the TIMESERIES clause, which stores the time slice start times generated from gap filling. <strong>Note:</strong> This parameter is an alias, so you can use any name that an alias would take.</td>
</tr>
<tr>
<td><code>length-and-time-unit-exp</code></td>
<td>An INTERVAL DAY TO SECOND literal that specifies the length of time unit of time slice computation. For example: TIME SERIES slice_time AS '3 seconds' ...</td>
</tr>
<tr>
<td><code>OVER()</code></td>
<td>Specifies partitioning and ordering for the function. OVER() also specifies that the time series function operates on a query result set—that is, the rows that are returned after the FROM, WHERE, GROUP BY, and HAVING clauses are evaluated.</td>
</tr>
<tr>
<td><code>PARTITION BY (column-expr</code></td>
<td>Partitions the data by the specified column expressions. <strong>Gap filling and interpolation</strong> is performed on each partition separately.</td>
</tr>
<tr>
<td><code>[,])</code></td>
<td></td>
</tr>
<tr>
<td><code>ORDER BY time-expr</code></td>
<td>Sorts the data by the TIMESTAMP expression <code>time-expr</code>, which computes the time information of the time series data. <strong>Note:</strong> The TIMESERIES clause requires an ORDER BY operation on the timestamp column.</td>
</tr>
</tbody>
</table>
Notes

If the window-partition-clause is not specified in TIMESERIES OVER(), for each defined time slice, exactly one output record is produced; otherwise, one output record is produced per partition per time slice. Interpolation is computed there.

Given a query block that contains a TIMESERIES clause, the following are the semantic phases of execution (after evaluating the FROM and the optional WHERE clauses):

1. Compute time-expression.

2. Perform the same computation as the TIME_SLICE() function on each input record based on the result of time-exp and 'length-and-time-unit-expr'.
   a. Perform gap filling to generate time slices missing from the input.
   b. Name the result of this computation as slice_time, which represents the generated “time series” column (alias) after gap filling.

3. Partition the data by expression, slice-time. For each partition, do step 4.

4. Sort the data by time-exp. Interpolation is computed here.

There is semantic overlap between the TIMESERIES clause and the TIME_SLICE function with the following key differences:

- TIMESERIES only supports the interval qualifier DAY TO SECOND; it does not allow YEAR TO MONTH.

- Unlike TIME_SLICE, the time slice length and time unit expressed in length-and-time-unit-expr must be constants so gaps in the time slices are well-defined.

- TIMESERIES performs gap filling; the TIME_SLICE function does not.

- TIME_SLICE can return the start or end time of a time slice, depending on the value of its fourth input parameter (start-or-end). TIMESERIES, on the other hand, always returns the start time of each time slice. To output the end time of each time slice, write a SELECT statement like the following:

  ```sql
  => SELECT slice_time + <slice_length>;
  ```
Restrictions

- When the TIMESERIES clause occurs in a SQL query block, only the following clauses can be used in the same query block:
  - SELECT
  - FROM
  - WHERE
  - ORDER BY

- GROUP BY and HAVING clauses are not allowed. If a GROUP BY operation is needed before or after gap-filling and interpolation (GFI), use a subquery and place the GROUP BY in the outer query. For example:

```sql
=> SELECT symbol, AVG(first_bid) as avg_bid FROM ( 
  SELECT symbol, slice_time, TS_FIRST_VALUE(bid1) AS first_bid 
  FROM Tickstore 
  WHERE symbol IN ('MSFT', 'IBM') 
  TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER BY ts) 
) AS resultOfGFI 
GROUP BY symbol;
```

- When the TIMESERIES clause is present in the SQL query block, the SELECT list can include only the following:
  - Time series aggregate functions such as `TS_FIRST_VALUE` and `TS_LAST_VALUE`
  - `slice_time` column
  - `PARTITION` BY expressions
  - `TIME_SLICE` function

For example, the following two queries return a syntax error because `bid1` is not a `PARTITION` BY or `GROUP` BY column:

```sql
=> SELECT bid, symbol, TS_FIRST_VALUE(bid) FROM Tickstore 
  TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER BY ts); 
ERROR: column "Tickstore.bid" must appear in the PARTITION BY list of Timeseries clause or be used in a Timeseries Output function
=> SELECT bid, symbol, AVG(bid) FROM Tickstore 
  GROUP BY symbol; 
ERROR: column "Tickstore.bid" must appear in the GROUP BY clause or be used in an aggregate function
```
Examples

For examples, see Gap Filling and Interpolation (GFI) in Analyzing Data.

See Also

- TIME_SLICE
- TS_FIRST_VALUE
- TS_LAST_VALUE
- Gap Filling and Interpolation (GFI)

UNION Clause

Combines the results of multiple SELECT statements. You can include UNION in FROM, WHERE, and HAVING clauses.

Syntax

```
select-stmt UNION { ALL | DISTINCT } select-stmt [ UNION { ALL | DISTINCT } select-stmt ]...
... [ ORDER BY expression { ASC | DESC }[,..] ]
... [ LIMIT { count | ALL } ]
... [ OFFSET start ]
```

Parameters

<table>
<thead>
<tr>
<th>select-stmt</th>
<th>A SELECT statement that returns one or more rows, depending on whether you specify keywords DISTINCT or ALL.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The following options also apply:</td>
</tr>
<tr>
<td></td>
<td>- The first SELECT statement can include the hint LABEL. Vertica ignores LABEL hints in subsequent SELECT statements.</td>
</tr>
<tr>
<td></td>
<td>- Each SELECT statement can specify its own ORDER BY, LIMIT, and OFFSET clauses. A SELECT statement with one or more of these clauses must be enclosed by parentheses. See also:</td>
</tr>
</tbody>
</table>
ORDER BY, LIMIT, and OFFSET Clauses in UNION

<table>
<thead>
<tr>
<th>DISTINCT</th>
<th>ALL</th>
<th>Specifies whether to return unique rows:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• DISTINCT (default) returns only unique rows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ALL concatenates all rows, including duplicates. For best performance, use UNION ALL.</td>
</tr>
</tbody>
</table>

Requirements

- All rows of the UNION result set must be in the result set of at least one of its SELECT statements.
- Each SELECT statement must specify the same number of columns.
- Data types of corresponding SELECT statement columns must be compatible, otherwise Vertica returns an error.

ORDER BY, LIMIT, and OFFSET Clauses in UNION

A UNION statement can specify its own ORDER BY, LIMIT, and OFFSET clauses. For example, given the tables described below in Examples, the following query orders the UNION result set by emp_name and limits output to the first two rows:

```
=> SELECT id, emp_name FROM company_a UNION ALL SELECT id, emp_name FROM company_b ORDER BY emp_name LIMIT 2;
  id | emp_name
-------
5678 | Alice
8765 | Bob
(2 rows)
```

Each SELECT statement in a UNION clause can specify its own ORDER BY, LIMIT, and OFFSET clauses. In this case, the SELECT statement must be enclosed by parentheses. Vertica processes the SELECT statement ORDER BY, LIMIT, and OFFSET clauses before it processes the UNION clauses.

For example, each SELECT statement in the following UNION specifies its own ORDER BY and LIMIT clauses. Vertica processes the individual queries and then concatenates the two result sets:

```
=> (SELECT id, emp_name FROM company_a ORDER BY emp_name LIMIT 2)
   UNION ALL
(SELECT id, emp_name FROM company_b ORDER BY emp_name LIMIT 2);
```
The following requirements and restrictions determine how Vertica processes a UNION clause that contains ORDER BY, LIMIT, and OFFSET clauses:

- A UNION's ORDER BY clause must specify columns from the first (leftmost) SELECT statement.
- Always use an ORDER BY clause with LIMIT and OFFSET. Otherwise, the query returns an undefined subset of the result set.
- ORDER BY must precede LIMIT and OFFSET.
- When a SELECT or UNION statement specifies both LIMIT and OFFSET, Vertica first processes the OFFSET statement, and then applies the LIMIT statement to the remaining rows.

**UNION in Non-Correlated Subqueries**

Vertica supports UNION in noncorrelated subquery predicates. For example:

```sql
=> SELECT DISTINCT customer_key, customer_name FROM public.customer_dimension WHERE customer_key IN
  (SELECT customer_key FROM store.store_sales_fact WHERE sales_dollar_amount > 500
  UNION
  SELECT customer_key FROM online_sales.online_sales_fact WHERE sales_dollar_amount > 500)
  AND customer_state = 'CT';
```

<table>
<thead>
<tr>
<th>customer_key</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7021</td>
<td>Luigi T. Dobisz</td>
</tr>
<tr>
<td>1971</td>
<td>Betty V. Dobisz</td>
</tr>
<tr>
<td>46284</td>
<td>Ben C. Gauthier</td>
</tr>
<tr>
<td>33885</td>
<td>Tanya Y. Taylor</td>
</tr>
<tr>
<td>5449</td>
<td>Sarah O. Robinson</td>
</tr>
<tr>
<td>29059</td>
<td>Sally Z. Fortin</td>
</tr>
<tr>
<td>11200</td>
<td>Foodhope</td>
</tr>
<tr>
<td>15582</td>
<td>John J. McNulty</td>
</tr>
<tr>
<td>24638</td>
<td>Alexandra F. Jones</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
```

**Examples**

The examples that follow use these two tables:
### company_a

<table>
<thead>
<tr>
<th>ID</th>
<th>emp_name</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>auto parts</td>
<td>1000</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
<td>auto parts</td>
<td>2500</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>floral</td>
<td>500</td>
</tr>
</tbody>
</table>

### company_b

<table>
<thead>
<tr>
<th>ID</th>
<th>emp_name</th>
<th>dept</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4321</td>
<td>Marvin</td>
<td>home goods</td>
<td>250</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>home goods</td>
<td>500</td>
</tr>
<tr>
<td>8765</td>
<td>Bob</td>
<td>electronics</td>
<td>20000</td>
</tr>
</tbody>
</table>

Find all employee IDs and names from company_a and company_b

The UNION statement specifies DISTINCT to combine unique IDs and last names of employees; Katherine works for both companies, so she appears only once in the result set. DISTINCT is the default and can be omitted:

```
=> SELECT id, emp_name FROM company_a UNION DISTINCT SELECT id, emp_name FROM company_b ORDER BY id;
```

```
+----------+------------+
<table>
<thead>
<tr>
<th>id</th>
<th>emp_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
</tr>
<tr>
<td>4321</td>
<td>Marvin</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
</tr>
<tr>
<td>8765</td>
<td>Bob</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
</tr>
</tbody>
</table>
(5 rows)
```

The next UNION statement specifies the option ALL. Katherine works for both companies, so the query returns two records for her:

```
=> SELECT id, emp_name FROM company_a UNION ALL SELECT id, emp_name FROM company_b ORDER BY id;
```

```
+----------+--------------------+
<table>
<thead>
<tr>
<th>id</th>
<th>emp_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
</tr>
<tr>
<td>4321</td>
<td>Marvin</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
</tr>
<tr>
<td>8765</td>
<td>Bob</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
</tr>
</tbody>
</table>
(6 rows)
```

Find the top two top performing salespeople in each company

Each SELECT statement specifies its own ORDER BY and LIMIT clauses, so the UNION statement concatenates the result sets as returned by each query:

```
=> (SELECT id, emp_name, sales FROM company_a ORDER BY sales DESC LIMIT 2) 
  UNION ALL 
  (SELECT id, emp_name, sales FROM company_b ORDER BY sales DESC LIMIT 2); 
```

```
+----------+----------+-------+
<table>
<thead>
<tr>
<th>id</th>
<th>emp_name</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>1000</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
<td>2500</td>
</tr>
<tr>
<td>4321</td>
<td>Marvin</td>
<td>250</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>500</td>
</tr>
</tbody>
</table>
(6 rows)
```
Find all employee orders by sales
The UNION statement specifies its own ORDER BY clause, which Vertica applies to the entire result:

```sql
 => SELECT id, emp_name, sales FROM company_a
      UNION
    SELECT id, emp_name, sales FROM company_b
    ORDER BY sales;
```

```
-+-+--------+
<table>
<thead>
<tr>
<th>id</th>
<th>emp_name</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>4321</td>
<td>Marvin</td>
<td>250</td>
</tr>
<tr>
<td>9012</td>
<td>Katherine</td>
<td>500</td>
</tr>
<tr>
<td>1234</td>
<td>Stephen</td>
<td>1000</td>
</tr>
<tr>
<td>5678</td>
<td>Alice</td>
<td>2500</td>
</tr>
<tr>
<td>8765</td>
<td>Bob</td>
<td>20000</td>
</tr>
</tbody>
</table>
(5 rows)
```

Calculate the sum of sales for each company grouped by department
Each SELECT statement has its own GROUP BY clause. UNION combines the aggregate results from each query:

```sql
 => (SELECT 'Company A' as company, dept, SUM(sales) FROM company_a
     GROUP BY dept)
     UNION
   (SELECT 'Company B' as company, dept, SUM(sales) FROM company_b
     GROUP BY dept)
     ORDER BY 1;
```

```
 company | dept       | sum  
---------|------------|------
 Company A | auto parts | 3500 |
 Company A | floral | 500 |
 Company B | electronics | 20000 |
 Company B | home goods | 750 |
(4 rows)
```

See Also

- SELECT
- EXCEPT Clause
• INTERSECT Clause
• Subqueries

WHERE Clause

Eliminates rows from the result table that do not satisfy one or more predicates.

Syntax

WHERE boolean-expression [ subquery ] ... 

Parameters

| boolean-expression | Is an expression that returns true or false. Only rows for which the expression is true become part of the result set. |

The boolean-expression can include Boolean Operators and the following elements:

• BETWEEN-predicate
• Boolean-Predicate
• Column-Value-Predicate
• IN-predicate
• Join-Predicate
• LIKE-predicate
• NULL-predicate

Notes

You can use parentheses to group expressions, predicates, and boolean operators. For example:

```sql
=> ... WHERE NOT (A=1 AND B=2) OR C=3;
```
Example

The following example returns the names of all customers in the Eastern region whose name starts with 'Amer'. Without the WHERE clause filter, the query returns all customer names in the customer_dimension table.

```sql
=> SELECT DISTINCT customer_name
    FROM customer_dimension
    WHERE customer_region = 'East'
    AND customer_name ILIKE 'Amer%';
```

<table>
<thead>
<tr>
<th>Americare</th>
<th>Americom</th>
<th>Americore</th>
<th>Americorp</th>
<th>Ameridata</th>
<th>Amerigen</th>
<th>Amerihope</th>
<th>Amerimedia</th>
<th>Amerishop</th>
<th>Ameristar</th>
<th>Ameritech</th>
</tr>
</thead>
</table>
(11 rows)

WITH Clause

WITH clauses are individually-evaluated SELECT statements for use in a larger container query. You can use WITH clauses to simplify complicated queries and reduce statement repetition.

WITH clauses are evaluated through inline expansion or (optionally) through materialization. For details, see WITH Clauses in SELECT.

Syntax

The following syntax statement is illustrative, rather than syntactically exact, to show the possibility of numerous successive WITH queries in use with others:

```sql
WITH... with-query-1 [(col-name[,,...])AS (SELECT ...),
... with-query-2 [(col-name[,,...])AS (SELECT ...[with-query-1]),
. .
... with-query-n [(col-name[,,...])AS (SELECT ...[with-query-1, with-query-2, with-query-n[,,...]])
SELECT
. .
```
Restrictions

- Each WITH clause query must be uniquely named. Same-name aliases for WITH clause query names return with an error.
- WITH clauses do not support INSERT, UPDATE, or DELETE statements.
- WITH clauses cannot be used recursively; they can only be specified in succession.

Examples

See WITH Clauses in SELECT.

See Also

- SELECT
- Subqueries
- WITH Clauses in SELECT

**SET DATESTYLE**

Specifies how to format date/time output for the current session. Use `SHOW DATESTYLE` to verify the current output settings.

Syntax

```
SET DATESTYLE TO { arg | 'arg' }[, arg | 'arg']
```

Parameters

`SET DATESTYLE` has a single parameter, which can be set to one or two arguments that specify date ordering and style. Each argument can be specified singly or in combination with the other; if combined, they can be specified in any order.
The following table describes each style and the date ordering arguments it supports:

<table>
<thead>
<tr>
<th>Date style arguments</th>
<th>Order arguments</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO (ISO 8601/SQL standard)</td>
<td>n/a</td>
<td>2016-03-16 00:00:00</td>
</tr>
<tr>
<td>GERMAN</td>
<td>n/a</td>
<td>16.03.2016 00:00:00</td>
</tr>
<tr>
<td>SQL</td>
<td>MDY (default)</td>
<td>03/16/2016 00:00:00</td>
</tr>
<tr>
<td></td>
<td>DMY</td>
<td>16/03/2016 00:00:00</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>MDY (default)</td>
<td>Wed Mar 16 00:00:00 2016</td>
</tr>
<tr>
<td></td>
<td>DMY</td>
<td>Wed 16 Mar 00:00:00 2016</td>
</tr>
</tbody>
</table>

Vertica ignores the order argument for date styles ISO and GERMAN. If the date style is SQL or POSTGRES, the order setting determines whether dates are output in MDY or DMY order. Neither SQL nor POSTGRES support YMD order. If you specify YMD for SQL or POSTGRES, Vertica ignores it and uses their default MDY order.

Date styles and ordering can also affect how Vertica interprets input values. For more information, see Date/Time Literals.

Privileges

None

Input Dependencies

In some cases, input format can determine output, regardless of date style and order settings:

- Vertica ISO output for DATEDSTYLE is ISO long form, but several input styles are accepted. If the year appears first in the input, YMD is used for input and output, regardless of the DATEDSTYLE value.

- INTERVAL input and output share the same format, with the following exceptions:
  - Units like CENTURY or WEEK are converted to years and days.
  - AGO is converted to the appropriate sign.

  If the date style is set to ISO, output follows this format:
Example

=> CREATE TABLE t(a DATETIME);
CREATE TABLE
=> INSERT INTO t values ('3/16/2016');
OUTPUT
--------
  1
(1 row)
=> SHOW DATESTYLE;
   name   | setting
-----------+----------
 datestyle | ISO, MDY
(1 row)

=> SELECT * FROM t;
a
-------------------
  2016-03-16 00:00:00
(1 row)

=> SET DATESTYLE TO German;
SET
=> SHOW DATESTYLE;
   name   | setting
-----------+----------
 datestyle | German, DMY
(1 row)

=> SELECT * FROM t;
a
-------------------
  16.03.2016 00:00:00
(1 row)

=> SET DATESTYLE TO SQL;
SET
=> SHOW DATESTYLE;
   name   | setting
-----------+----------
 datestyle | SQL, DMY
(1 row)

=> SELECT * FROM t;
a
-------------------
  16/03/2016 00:00:00
(1 row)

=> SET DATESTYLE TO Postgres, MDY;
SET
=> SHOW DATESTYLE;
### SET ESCAPE_STRING_WARNING

Issues a warning when a backslash is used in a string literal during the current session.

#### Syntax

```
SET ESCAPE_STRING_WARNING TO { ON | OFF }
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| ON        | [Default] Issues a warning when a back slash is used in a string literal.  
            Tip: Organizations that have upgraded from earlier versions of Vertica can use this as a debugging tool for locating backslashes that used to be treated as escape characters, but are now treated as literals. |
| OFF       | Ignores back slashes within string literals. |

#### Privileges

No special permissions required.

#### Notes

- This statement works under vsql only.
- Turn off standard conforming strings before you turn on this parameter.
Tip: To set escape string warnings across all sessions, use the EscapeStringWarnings configuration parameter. See the Internationalization Parameters in the Administrator's Guide.

Examples

The following example shows how to turn OFF escape string warnings for the session.

```sql
=> SET ESCAPE_STRING_WARNING TO OFF;
```

See Also

- `SET STANDARD_CONFORMING_STRINGS`

SET INTERVALSTYLE

Specifies whether to include units in interval output for the current session.

Syntax

```sql
SET INTERVALSTYLE TO [ plain | units ]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>Sets the default interval output to omit units. PLAIN is the default value.</td>
</tr>
<tr>
<td>units</td>
<td>Enables interval output to include subtype unit identifiers. When INTERVALSTYLE is set to units, the DATESTYLE parameter controls output. If you enable units and they do not display in the output, check the DATESTYLE parameter value, which must be set to ISO or POSTGRES for interval units to display.</td>
</tr>
</tbody>
</table>

Privileges

None
Examples

See Setting Interval Unit Display.

**SET LOCALE**

Specifies locale for the current session.

**Syntax**

```sql
SET LOCALE TO ICU-locale-identifier
```

**Parameters**

| **ICU-locale-identifier** | Specifies the ICU locale identifier to use, by default set to:  
en_US@collation=binary  
You can also use the vsqI command `\locale` to set the current locale. An unqualified `\locale` command shows the current setting: |
|---------------------------|---------------------------------------------------------------|
|                           | => `\locale`  
en_GB  
=> `\locale` en_US@collation=binary;  
INFO 2567: Canonical locale: 'en_US'  
Standard collation: 'LEN_KBINARY'  
English (United States)  
=> `\locale`  
en_US@collation=binary; |

If set to null, Vertica sets locale to en_US_POSIX:

```sql
=> set locale to '';  
INFO 2567: Canonical locale: 'en_US_POSIX'  
Standard collation: 'LEN'  
English (United States, Computer)  
SET
```

The following requirements apply:

- Vertica only supports the COLLATION keyword.
Single quotes are mandatory to specify collation.

For a complete list of locale identifiers, see the ICU Project

Privileges

None

Commonly Used Locales

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>de_DE</td>
<td>German (Germany)</td>
</tr>
<tr>
<td>en_GB</td>
<td>English (Great Britain)</td>
</tr>
<tr>
<td>es_ES</td>
<td>Spanish (Spain)</td>
</tr>
<tr>
<td>fr_FR</td>
<td>French (France)</td>
</tr>
<tr>
<td>pt_BR</td>
<td>Portuguese (Brazil)</td>
</tr>
<tr>
<td>pt_PT</td>
<td>Portuguese (Portugal)</td>
</tr>
<tr>
<td>ru_RU</td>
<td>Russian (Russia)</td>
</tr>
<tr>
<td>ja_JP</td>
<td>Japanese (Japan)</td>
</tr>
<tr>
<td>zh_CN</td>
<td>Chinese (China, simplified Han)</td>
</tr>
<tr>
<td>zh_Hant_TW</td>
<td>Chinese (Taiwan, traditional Han)</td>
</tr>
</tbody>
</table>

Examples

Set session locale to en_GB:

```sql
=> SET LOCALE TO en_GB;
INFO 2567: Canonical locale: 'en_GB'
Standard collation: 'LEN'
English (United Kingdom)
SET
```

Use the short form of a locale:
=> SET LOCALE TO LEN;
INFO 2567: Canonical locale: 'en'
Standard collation: 'LEN'
English
SET

Specify collation:

=> SET LOCALE TO 'tr_tr@collation=standard';
INFO 2567: Canonical locale: 'tr_TR@collation=standard'
Standard collation: 'LTR'
Turkish (Turkey, collation=standard) Türkçe (Türkiye, Siralama=standard)
SET

See Also

- Implement Locales for International Data Sets
- About Locale

SET ROLE

Enables a role for the user's current session. The user can access privileges that have been granted to the role.

**Note:** If you set EnableAllRolesOnLogin=1, this eliminates the need for the user to run SET ROLE <rolenames> to enable any roles that have been granted to any user. For more information see Security Parameters.

Syntax

\[
\text{SET ROLE} \ {role \ [, \ \ldots] \ | \ NONE \ | \ ALL \ | \ ALL \ EXCEPT \ [, \ \ldots] \ | \ DEFAULT}
\]

Parameters

\[
\begin{array}{c|c}
\text{role} \ [, \ \ldots] \ | \ NONE \ | \ ALL \ | \ ALL \ EXCEPT \ [, \ \ldots] \ | \ DEFAULT & \text{The name of one or more roles to set as the current role, or one of the following keywords:} \\
\end{array}
\]
- NONE disables all roles for the current user session.
- ALL enables all of the roles to which the user has been granted access. Use `GRANT (Role)` to assign a role to a user.
- ALL EXCEPT enable all the roles to which the user has access, with the exception of the role or roles indicated with this command.
- DEFAULT sets the roles assigned to the user as the default roles.

### Privileges

As a user, you can only set a role that has been granted to you. Use the `SHOW AVAILABLE ROLES` command to retrieve a list of the roles available to you.

### Notes

- The DBADMIN user creates default roles that the DBADMIN then grants to a user. The DBADMIN can also grant a default role to a user using `ALTER USER`.

- Enabling a role does not affect any other roles that are currently enabled. A user session can have more than one role enabled at a time. The user's permissions are the union of all the roles that are currently active, plus any permissions granted directly to the user.

### Examples

This example shows the following:

- `SHOW AVAILABLE_ROLES`; lists the roles available to the user, but not enabled.
- `SET ROLE applogs`; enables the applogs role for the user.
- `SHOW ENABLED_ROLES`; lists the applogs role as enabled (SET) for the user.
- SET ROLE appuser; enables the appuser role for the user.

- SHOW ENABLED_ROLES now lists both applogs and appuser as enabled roles for the user.

- SET ROLE NONE disables all the users' enabled roles.

- SHOW ENABLED_ROLES shows that no roles are enabled for the user.

```
=> SHOW AVAILABLE_ROLES;
+----------------+---------+
| name           | setting |
|----------------+---------|
| available roles | applogs, appadmin, appuser |
+----------------+---------+
(1 row)
=> SET ROLE applogs;
SET

=> SHOW ENABLED_ROLES;
+----------------+---------+
| name           | setting |
|----------------+---------|
| enabled roles   | applogs  |
+----------------+---------+
(1 row)

=> SET ROLE appuser;
SET

=> SHOW ENABLED_ROLES;
+----------------+---------+
| name           | setting |
|----------------+---------|
| enabled roles   | applogs, appuser |
+----------------+---------+
(1 row)

=> SET ROLE NONE;
SET

=> SHOW ENABLED_ROLES;
+----------------+---------+
| name           | setting |
|----------------+---------|
| enabled roles   |         |
+----------------+---------+
(1 row)
```

**Set User Default Roles**

Though the DBADMIN user is normally responsible for setting a user's default roles, as a user you can set your own role. For example, if you run SET ROLE NONE all of your enabled roles are disabled. Then it was determined you need access to role1 as a default role. The DBADMIN uses **ALTER USER** to assign you a default role:

```
=> ALTER USER user1 default role role1;
```

This example sets role1 as user1's default role because the DBADMIN assigned this default role using ALTER USER.
Set All Roles as Default

This example makes all roles granted to user1 default roles:

```sql
user1 => SET ROLE default;
user1 => SHOW ENABLED_ROLES;
    name  | setting
---------------------
enabled roles | role1
(1 row)
```

Set All Roles as Default With EXCEPT

This example makes all the roles granted to the user default roles with the exception of role1.

```sql
user1 => set role all except role1;
user1 => SHOW ENABLED_ROLES
    name  | setting
---------------------
enabled roles | role2, role3
(1 row)
```

SET SEARCH_PATH

Specifies the order in which Vertica searches schemas when a SQL statement specifies a table name that is unqualified by a schema name. SET SEARCH_PATH overrides the current session’s search path, which is initially set from the user profile. This search path remains in effect until the next SET SEARCH_PATH statement, or the session ends. For details, see Setting Search Paths in the Administrator’s Guide.

To view the current search path, use SHOW SEARCH_PATH.

Syntax

```sql
SET SEARCH_PATH { TO | = } { schema-list | DEFAULT }
```
Parameters

<table>
<thead>
<tr>
<th>schema-list</th>
<th>A comma-delimited list of schemas that indicates the order in which Vertica searches schemas for a table whose name is unqualified by a schema name. If the search path includes a schema that does not exist, or for which the user lacks access privileges, Vertica silently skips over that schema.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>Sets the search path to the database default: &quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
</tbody>
</table>

Privileges

None

Examples

Show the current search path:

```
=> SHOW SEARCH_PATH;
 name | setting
-----------------------
 search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

Reset the search path to schemas store and public:

```
=> SET SEARCH_PATH TO store, public;
 => SHOW SEARCH_PATH;
 name | setting
-----------------------
 search_path | store, public, v_catalog, v_monitor, v_internal
(1 row)
```

Reset the search path to the database default settings:

```
=> SET SEARCH_PATH TO DEFAULT;
 SET
 => SHOW SEARCH_PATH;
 name | setting
-----------------------
 search_path | "$user", public, v_catalog, v_monitor, v_internal
```
**SET SESSION AUTOCOMMIT**

Sets whether statements automatically commit their transactions on completion. This statement is primarily used by the client drivers to enable and disable autocommit, you should never have to directly call it.

**Syntax**

```sql
SET SESSION AUTOCOMMIT TO { ON | OFF }
```

**Parameters**

<table>
<thead>
<tr>
<th>ON</th>
<th>Enable autocommit. Statements automatically commit their transactions when they complete. This is the default setting for connections made using the Vertica client libraries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Disable autocommit. Transactions are not automatically committed. This is the default for interactive sessions (connections made through vsql).</td>
</tr>
</tbody>
</table>

**Privileges**

No special permissions required.

**Examples**

This examples show how to set AUTOCOMMIT to 'on' and then to 'off'.

```
=> SET SESSION AUTOCOMMIT TO on;
SET
=> SET SESSION AUTOCOMMIT TO off;
SET
```
See Also

- Client Libraries

SET SESSION CHARACTERISTICS AS TRANSACTION

Sets the isolation level and access mode of all transactions that start after this statement is issued.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

Syntax

```
SET SESSION CHARACTERISTICS AS TRANSACTION setting[, setting]
```

`setting` = one or both of the following:

- `ISOLATION LEVEL argument`
- `READ ONLY | READ WRITE`

ISOLATION LEVEL Arguments

The `ISOLATION LEVEL` clause determines what data the transaction can access when other transactions run concurrently. You cannot change the isolation level after the first query (SELECT) or DML statement (INSERT, DELETE, UPDATE) if a transaction has run.

Set `ISOLATION LEVEL` to one of the following arguments:

<table>
<thead>
<tr>
<th>SERIALIZABLE</th>
<th>Sets the strictest level of SQL transaction isolation. This level emulates transactions serially, rather than concurrently. It holds locks and blocks write operations until the transaction completes.</th>
</tr>
</thead>
</table>
Applications that use SERIALIZABLE must be prepared to retry transactions in the event of serialization failures. This isolation level is not recommended for normal query operations.

Setting the transaction isolation level to SERIALIZABLE does not apply to temporary tables. Temporary tables are isolated by their transaction scope.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPEATABLE READ</td>
<td>Automatically converted to SERIALIZABLE.</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>The default setting, allows concurrent transactions.</td>
</tr>
<tr>
<td>READ UNCOMMITTED</td>
<td>Automatically converted to READ COMMITTED.</td>
</tr>
</tbody>
</table>

**READ WRITE/READ ONLY**

You can set the transaction access mode with one of the following:

<table>
<thead>
<tr>
<th>Access Mode</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ WRITE</td>
<td>Default</td>
</tr>
<tr>
<td>READ ONLY</td>
<td>Disallows SQL statements that require write access:</td>
</tr>
<tr>
<td></td>
<td>• INSERT, UPDATE, DELETE, and COPY operations on any non-temporary table.</td>
</tr>
<tr>
<td></td>
<td>• CREATE, ALTER, and DROP</td>
</tr>
<tr>
<td></td>
<td>• GRANT, REVOKE</td>
</tr>
<tr>
<td></td>
<td>• EXPLAIN if the SQL statement to explain requires write access.</td>
</tr>
</tbody>
</table>

**Note:** Setting the transaction session mode to read-only does not prevent all write operations.

**Privileges**

None
Viewing Session Transaction Characteristics

SHOW TRANSACTION_ISOLATION and SHOW TRANSACTION_READ_ONLY show the transaction settings for the current session:

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_isolation</td>
<td>SERIALIZABLE</td>
</tr>
</tbody>
</table>

(1 row)

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_read_only</td>
<td>true</td>
</tr>
</tbody>
</table>

(1 row)

**SET SESSION GRACEPERIOD**

Sets how long a session socket remains blocked while awaiting client input or output for a given query. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated. If no grace period is set, the query can maintain its block on the socket indefinitely.

Vertica applies a session's grace period and RUNTIMECAP settings independently. If no grace period is set, a query can continue to block indefinitely on a session socket, regardless of the query's RUNTIMECAP setting.

**Syntax**

`SET SESSION GRACEPERIOD duration`

**Parameters**

<table>
<thead>
<tr>
<th>duration</th>
<th>Specifies how long a query can block on any session socket, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>'interval': Specifies as an interval the maximum grace period for current session queries, up to 20 days.</td>
</tr>
</tbody>
</table>
Privileges

- Superusers can increase session grace period to any value, regardless of database or node settings.

- Non-superusers can only set the session grace period to a value equal to or lower than their own user setting. If no grace period is explicitly set for a user, the grace period for that user is inherited from the node or database settings.

Examples

See Handling Session Socket Blocking in the Administrator's Guide.

**SET SESSION IDLESESSIONTIMEOUT**

Sets the maximum amount of time that a session can remain idle before it exits.

**Note:** An idle session has no queries running.

Syntax

```
SET SESSION IDLESESSIONTIMEOUT duration
```

Parameters

<table>
<thead>
<tr>
<th><strong>duration</strong></th>
<th>Specifies the amount of time a session can remain idle before it exits:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NONE</strong> (default): No idle timeout set on the session.</td>
<td></td>
</tr>
</tbody>
</table>
`interval`: Specifies as an `interval` the maximum amount of time a session can remain idle.

`=DEFAULT`: Sets the idle timeout period for this session to the user's `IDLESESSIONTIMEOUT` value.

### Privileges

- Superusers can increase the time a session can remain idle to any value, regardless of database or node settings.

- Non-superusers can only set the session idle time to a value equal to or lower than their own user setting. If no session idle time is explicitly set for a user, the session idle time for that user is inherited from the node or database settings.

### Examples

See [Managing Client Connections](#) in the Administrator's Guide.

### `SET SESSION MEMORYCAP`

Limits how much memory can be allocated to any request in this session. This limit only applies to the current session; it does not limit the total amount of memory used by multiple sessions.

#### Syntax

```
SET SESSION MEMORYCAP limit
```

#### Parameters

<table>
<thead>
<tr>
<th><code>limit</code></th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>max-expression</code>: A string value that specifies the memory limit, one of the following:</td>
</tr>
</tbody>
</table>
- `int%` — Expresses the maximum as a percentage of total memory available to the Resource Manager, where `int` is an integer value between 0 and 100. For example:

  \[
  \text{MEMORYCAP '40%'}
  \]

- `int{K|M|G|T}` — Expresses memory allocation in kilobytes, megabytes, gigabytes, or terabytes. For example:

  \[
  \text{MEMORYCAP '10G'}
  \]

- `=DEFAULT`: Sets the memory cap for queries in this session to the user's MEMORYCAP value. A new session is initially set to this value.

- `NONE`: Valid only for superusers, removes any grace period previously set on session queries.

### Privileges

- Superusers can increase session memory cap to any value.

- Non-superusers can only set the session memory cap to a value equal to or lower than their own user setting.

### Examples

**Set the session memory cap to 2 gigabytes:**

```sql
=> SET SESSION MEMORYCAP '2G';
SET
=> SHOW MEMORYCAP;
  name  | setting
----------
 memorycap | 2097152
(1 row)
```

**Revert the memory cap to the default setting as specified in the user profile:**

```sql
=> SET MEMORYCAP=DEFAULT;
SET
=> SHOW MEMORYCAP;
  name  | setting
----------
 memorycap | 2013336
```
See Also

Managing Workloads

**SET SESSION MULTIPLEACTIVERESULTSETS**

Enables or disable the execution of multiple active result sets (MARS) on a single JDBC connection. Using this option requires an active JDBC connection.

**Syntax**

```
SET SESSION MULTIPLEACTIVERESULTSETS TO {ON | OFF}
```

**Parameters**

<table>
<thead>
<tr>
<th>ON</th>
<th>Enable MultipleActiveResultSets. Allows you to execute multiple result sets on a single connection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Disable MultipleActiveResultSets. Allows only one active result set per connection. (Default value.)</td>
</tr>
</tbody>
</table>

**Privileges**

No special permissions required.

**Examples**

This example shows how you can set MultipleActiveResultSets to on and then to off:
SET SESSION RESOURCE_POOL

Associates the user session with the specified resource pool.

Syntax

SET SESSION RESOURCE_POOL = { pool-name | DEFAULT }

Parameters

<table>
<thead>
<tr>
<th>pool-name</th>
<th>The name of an existing resource pool to associate with the current session.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>Sets the session's resource pool to the user's default resource pool.</td>
</tr>
</tbody>
</table>

Privileges

- Superusers can assign their session to any available resource pool.
- Non-superusers must have USAGE privileges for the resource pool.

Examples

This example sets ceo_pool as the session resource pool:

```sql
=> SET SESSION RESOURCE_POOL = ceo_pool;
SET
```
See Also

- ALTER RESOURCE POOL
- CREATE RESOURCE POOL
- CREATE USER
- DROP RESOURCE POOL
- GRANT (Resource Pool)
- SET SESSION MEMORYCAP
- Managing Workloads

**SET SESSION RUNTIMECAP**

Sets the maximum amount of time queries can run in a given session. If a query exceeds its session's RUNTIMECAP setting, Vertica terminates the query and returns an error. You cannot increase the RUNTIMECAP beyond the limit that is set in your user profile.

Note: Vertica does not strictly enforce session RUNTIMECAP settings. If you time a query, you might discover that it runs longer than the RUNTIMECAP setting.

**Syntax**

```
SET SESSION RUNTIMECAP duration
```

**Parameters**

<table>
<thead>
<tr>
<th>duration</th>
<th>Specifies how long a given query can run in the current session, one of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• NONE (default): Removes a runtime limit for all current session queries.</td>
</tr>
<tr>
<td></td>
<td>• 'interval': Specifies as an interval the maximum runtime for current</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
session queries, up to one year—for example, 1 minute or 100 seconds.

- =DEFAULT: Sets maximum runtime for queries in this session to the user's RUNTIMECAP value.

Privileges

- Superusers can increase session RUNTIMECAP to any value.
- Non-superusers can only set the session RUNTIMECAP to a value equal to or lower than their own user RUNTIMECAP.

Examples

Set the maximum query runtime for the current session to 10 minutes:

```
=> SET SESSION RUNTIMECAP '10 minutes';
```

Revert the session RUNTIMECAP to your user default setting:

```
=> SET SESSION RUNTIMECAP =DEFAULT;
SET
=> SHOW RUNTIMECAP;
    name     | setting
--------------------------
    runtimecap | UNLIMITED
(1 row)
```

See Also

- Setting a Runtime Limit for Queries
- Managing Workloads

**SET SESSION TEMPSPACECAP**

Sets the maximum amount of temporary file storage space that any request issued by the session can consume.
Syntax

```
SET SESSION TEMPSPACECAP 'space-limit' = default | NONE
```

Parameters

| 'space-limit' | The maximum amount of temporary file space the session can use. To set a limit, use a numeric value followed by a unit (for example: '10G'). The unit can be one of the following:
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
|   | % percentage of total temporary storage space available. (In this case, the numeric value must be 0-100).
|   | K—Kilobytes
|   | M—Megabytes
|   | G—Gigabytes
|   | T—Terabytes

Setting this value to `default` sets the session's TEMPSPACECAP to the user's TEMPSPACECAP value.

Setting this value to `NONE` results in the session having unlimited temporary storage space. This is the default value.

Privileges

- This command requires superuser privileges to increase the TEMPSPACECAP over the user's TEMPSPACECAP limit.

- Regular users can change the TEMPSPACECAP associated with their own sessions to any value less than or equal to their own TEMPSPACECAP. They cannot increase its value beyond their own TEMPSPACECAP value.
Notes

- This limit is per session, not per user. A user could open multiple sessions, each of which could use up to the TEMPSPACECAP.

- Any execution plan that exceeds its TEMPSPACECAP usage results in the error:

  ERROR: Exceeded temp space cap.

Examples

The following command sets a TEMPSPACECAP of 20gigabytes on the session:

```sql
=> SET SESSION TEMPSPACECAP '20G';
SET
=> SHOW TEMPSPACECAP;
  name   | setting
--------------------------------
  tempspacecap | 20971520
(1 row)
```

**Note**: SHOW displays the TEMPSPACECAP in kilobytes.

To return the memorycap to the previous setting:

```sql
=> SET SESSION TEMPSPACECAP NONE;
SET
=> SHOW TEMPSPACECAP;
  name   | setting
--------------------------------
  tempspacecap | UNLIMITED
(1 row)
```

See Also

- ALTER USER
- CREATE USER
- Managing Workloads
SET STANDARD_CONFORMING_STRINGS

Treats backslashes as escape characters for the current session.

Syntax

SET STANDARD_CONFORMING_STRINGS TO { ON | OFF }

Parameters

| ON | Makes ordinary string literals ('...') treat back slashes (\) literally. This means that back slashes are treated as string literals, not escape characters. (This is the default.) |
| OFF | Treats back slashes as escape characters. |

Privileges

No special permissions required.

Notes

- This statement works under vsql only.
- When standard conforming strings are on, Vertica supports SQL:2008 string literals within Unicode escapes.
- Standard conforming strings must be ON to use Unicode-style string literals (U' \nnnn ').

Tip: To set conforming strings across all sessions (permanently), use the StandardConformingStrings as described in Internationalization Parameters in the Administrator's Guide.
Examples

The following example shows how to turn off conforming strings for the session.

```sql
=> SET STANDARD_CONFORMING_STRINGS TO OFF;
```

The following command lets you verify the settings:

```sql
=> SHOW STANDARD_CONFORMING_STRINGS;

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard_conforming_strings</td>
<td>off</td>
</tr>
</tbody>
</table>
```

The following example shows how to turn on conforming strings for the session.

```sql
=> SET STANDARD_CONFORMING_STRINGS TO ON;
```

See Also

- `SET ESCAPE_STRING_WARNING`

SET TIME ZONE

Changes the TIME ZONE run-time parameter for the current session. Use `SHOW TIMEZONE` to show the session's current time zone.

If you set the timezone using POSIX format, the timezone abbreviation you use overrides the default timezone abbreviation. If the `date style` is set to POSTGRES, the timezone abbreviation you use is also used when converting a timestamp to a string.

Syntax

```sql
SET TIME ZONE TO { value | 'value' }
```

Note: Vertica treats literals `TIME ZONE` and `TIMEZONE` as synonyms.
Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>One of the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A time zone literal supported by Vertica. To view the default list of valid literals, see the files in the following directory: opt/vertica/share/timezonesets</td>
</tr>
<tr>
<td></td>
<td>A signed integer representing an offset from UTC in hours</td>
</tr>
<tr>
<td></td>
<td>An interval value</td>
</tr>
<tr>
<td></td>
<td>Constants LOCAL and DEFAULT, which respectively set the time zone to the one specified in environment variable TZ, or if TZ is undefined, to the operating system time zone.</td>
</tr>
</tbody>
</table>

Privileges

None

Examples

=> SET TIME ZONE TO DEFAULT;
=> SET TIME ZONE TO 'PST8PDT'; -- Berkeley, California
=> SET TIME ZONE TO 'Europe/Rome'; -- Italy
=> SET TIME ZONE TO '-7'; -- UDT offset equivalent to PDT
=> SET TIME ZONE TO INTERVAL '-08:00 HOURS';

See Also

Using Time Zones With Vertica

Time Zone Names for Setting TIME ZONE

The following time zone names are recognized by Vertica as valid settings for the SQL time zone (the TIME ZONE run-time parameter).
Note: The names listed here are for convenience only and might be out of date. Refer to the Sources for Time Zone and Daylight Saving Time Data page for precise information.

These names are not the same as the names shown in /opt/vertica/share/timezonesets, which are recognized by Vertica in date/time input values. The TIME ZONE names shown below imply a local daylight-savings time rule, where date/time input names represent a fixed offset from UTC.

In many cases the same zone has several names. These are grouped together. The table is primarily sorted by commonly used zone names.

In addition to the names listed in the table, Vertica accepts time zone names of the form STDoffset or STDoffsetDST, where STD is a zone abbreviation, offset is a numeric offset in hours west from UTC, and DST is an optional daylight-savings zone abbreviation, assumed to stand for one hour ahead of the given offset. For example, if EST5EDT were not already a recognized zone name, it would be accepted and would be functionally equivalent to USA East Coast time. When a daylight-savings zone name is present, it is assumed to be used according to USA time zone rules, so this feature is of limited use outside North America. Be wary that this provision can lead to silently accepting bogus input, since there is no check on the reasonableness of the zone abbreviations. For example, SET TIME ZONE TO FOOBANKO works, leaving the system effectively using a rather peculiar abbreviation for GMT.

<table>
<thead>
<tr>
<th>Time Zone</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Africa</td>
<td></td>
</tr>
<tr>
<td>America</td>
<td></td>
</tr>
<tr>
<td>Antarctica</td>
<td></td>
</tr>
<tr>
<td>Asia</td>
<td></td>
</tr>
<tr>
<td>Atlantic</td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td></td>
</tr>
<tr>
<td>CET</td>
<td></td>
</tr>
<tr>
<td>EET</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT</td>
<td></td>
</tr>
<tr>
<td>Europe</td>
<td></td>
</tr>
</tbody>
</table>
### Time Zone

<table>
<thead>
<tr>
<th>Factory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Etc/GMT</td>
</tr>
<tr>
<td>GMT</td>
</tr>
<tr>
<td>GMT+0</td>
</tr>
<tr>
<td>GMT-0</td>
</tr>
<tr>
<td>GMT0</td>
</tr>
<tr>
<td>Greenwich</td>
</tr>
<tr>
<td>Etc/Greenwich</td>
</tr>
<tr>
<td>Indian</td>
</tr>
<tr>
<td>MET</td>
</tr>
<tr>
<td>Pacific</td>
</tr>
<tr>
<td>UCT Etc</td>
</tr>
<tr>
<td>UCT</td>
</tr>
<tr>
<td>UTC</td>
</tr>
<tr>
<td>Universal Zulu</td>
</tr>
<tr>
<td>Etc/UTC</td>
</tr>
<tr>
<td>Etc/Universal</td>
</tr>
<tr>
<td>Etc/Zulu</td>
</tr>
<tr>
<td>WET</td>
</tr>
</tbody>
</table>

### SHOW

Displays run-time parameters for the current session.

**Syntax**

```
SHOW { parameter | ALL }
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTOCOMMIT</strong></td>
<td>Displays whether statements automatically commit their transactions when they complete.</td>
</tr>
<tr>
<td><strong>AVAILABLE_ROLES</strong></td>
<td>Lists all roles available to the user.</td>
</tr>
<tr>
<td><strong>DATESTYLE</strong></td>
<td>Displays the current style of date values. See <a href="#">SET DATESTYLE</a>.</td>
</tr>
<tr>
<td><strong>ENABLED_ROLES</strong></td>
<td>Displays the roles enabled for the current session. See <a href="#">SET ROLE</a>.</td>
</tr>
<tr>
<td><strong>ESCAPE_STRING_WARNING</strong></td>
<td>Displays whether warnings are issued when backslash escapes are found in strings. See <a href="#">SET ESCAPE_STRING_WARNING</a>.</td>
</tr>
<tr>
<td><strong>INTERVALSTYLE</strong></td>
<td>Displays whether units are output when printing intervals. See <a href="#">SET INTERVALSTYLE</a>.</td>
</tr>
<tr>
<td><strong>LOCALE</strong></td>
<td>Displays the current locale. See <a href="#">SET LOCALE</a>.</td>
</tr>
<tr>
<td><strong>MEMORYCAP</strong></td>
<td>Displays the maximum amount of memory that any request use. See <a href="#">SET MEMORYCAP</a>.</td>
</tr>
<tr>
<td><strong>MULTIPLEACTIVERESULTSETS</strong></td>
<td>Displays whether multiple active result sets on one connection are allowed. See <a href="#">SET SESSION MULTIPLEACTIVERESULTSETS</a>.</td>
</tr>
<tr>
<td><strong>RESOURCE_POOL</strong></td>
<td>Displays the resource pool that the session is using. See <a href="#">SET RESOURCE_POOL</a>.</td>
</tr>
<tr>
<td><strong>RUNTIMECAP</strong></td>
<td>Displays the maximum amount of time that queries can run in the session. See <a href="#">SET RUNTIMECAP</a>.</td>
</tr>
<tr>
<td><strong>SEARCH_PATH</strong></td>
<td>Displays the order in which Vertica searches schemas. See <a href="#">SET SEARCH_PATH</a>.</td>
</tr>
<tr>
<td><strong>STANDARD_CONFORMING_STRINGS</strong></td>
<td>Displays whether backslash escapes are enabled for the session. See <a href="#">SET STANDARD_CONFORMING_STRINGS</a>.</td>
</tr>
<tr>
<td><strong>TEMPSPACECAP</strong></td>
<td>Displays the maximum amount of temporary file space that queries can use in the session. See <a href="#">SET</a>.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIMEZONE</td>
<td>Displays the timezone set in the current session. See <code>SET TIMEZONE</code>.</td>
</tr>
<tr>
<td>TRANSACTION_ISOLATION</td>
<td>Displays the current transaction isolation setting, as described in <code>SET SESSION CHARACTERISTICS AS TRANSACTION</code>.</td>
</tr>
<tr>
<td>TRANSACTION_READ_ONLY</td>
<td>Displays the current read-only setting, as described in <code>SET SESSION CHARACTERISTICS AS TRANSACTION</code>.</td>
</tr>
<tr>
<td>ALL</td>
<td>Shows all run-time settings.</td>
</tr>
</tbody>
</table>

**Privileges**

None

**Examples**

Display all current runtime parameter settings:

```
=> SHOW ALL;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>en_US@collation=binary (LEN_KBINARY)</td>
</tr>
<tr>
<td>autocommit</td>
<td>off</td>
</tr>
<tr>
<td>standard_conforming_strings</td>
<td>on</td>
</tr>
<tr>
<td>escape_string_warning</td>
<td>on</td>
</tr>
<tr>
<td>datestyle</td>
<td>ISO, MDY</td>
</tr>
<tr>
<td>intervalstyle</td>
<td>plain</td>
</tr>
<tr>
<td>timezone</td>
<td>US/Eastern</td>
</tr>
<tr>
<td>search_path</td>
<td>&quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
<tr>
<td>transaction_isolation</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>transaction_read_only</td>
<td>false</td>
</tr>
<tr>
<td>resource_pool</td>
<td>general</td>
</tr>
<tr>
<td>memorycap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>tempspacecap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>runtimecap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>enabled roles</td>
<td></td>
</tr>
<tr>
<td>available roles</td>
<td>applogs, appadmin</td>
</tr>
</tbody>
</table>

(15 rows)

Return current search path settings:

```
=> SHOW SEARCH_PATH;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
</table>

Vertica Analytic Database (9.0.x)
Show the session's transaction isolation level:

```sql
=> SHOW TRANSACTION_ISOLATION;
   name           | setting
-----------------+---------------------
  transaction_isolation | READ COMMITTED
(1 row)
```

Return the current transaction isolation level, as set by `SET SESSION CHARACTERISTICS AS TRANSACTION`. False indicates that the default read/write setting is in effect:

```sql
=> SHOW TRANSACTION_READ_ONLY;
   name        | setting
-------------+----------
  transaction_read_only | false
(1 row)
```

To change to read only:

```sql
=> SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY;
```

The same `SHOW` command now returns true:

```sql
=> SHOW TRANSACTION_READ_ONLY;
   name        | setting
-------------+----------
  transaction_read_only | true
(1 row)
```

**SHOW CURRENT**

Displays active configuration parameter values that are set at all levels. Vertica first checks values set at the session level. If a value is not set for a configuration parameter at the session level, Vertica next checks if the value is set for the node where you are logged in, and then checks the database level. If no values are set, `SHOW CURRENT` shows the default value for the configuration parameter. If the configuration parameter requires a restart to take effect, the active values shown might differ from the set values.

**Syntax**

```
SHOW CURRENT { parameter-name[,...] | ALL }
```
Parameters

<table>
<thead>
<tr>
<th>parameter-name</th>
<th>Names of any configuration parameters you want to show.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Shows all configuration parameters set at all levels.</td>
</tr>
</tbody>
</table>

Privileges

- Superusers only


Examples

Show configuration parameters and their settings at all levels.

```
=> SHOW CURRENT ALL;
```

<table>
<thead>
<tr>
<th>level</th>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>ActivePartitionCount</td>
<td>1</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>AdvanceAHMInterval</td>
<td>180</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>AHMBackupManagement</td>
<td>0</td>
</tr>
<tr>
<td>DATABASE</td>
<td>AnalyzeRowCountInterval</td>
<td>3600</td>
</tr>
<tr>
<td>SESSION</td>
<td>ForceUDxFencedMode</td>
<td>1</td>
</tr>
<tr>
<td>NODE</td>
<td>MaxClientSessions</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SHOW DATABASE

Displays configuration parameter values that are set for the database.

Note: If the configuration parameter is set but requires a database restart to take effect, the value shown might differ from the active value.
Syntax

SHOW DATABASE  

Parameters

<table>
<thead>
<tr>
<th><strong>db-name</strong></th>
<th>Specifies the current database, set to the database name or DEFAULT.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parameter-name</strong></td>
<td>Names of one or more configuration parameters to show. Non-superusers can only specify parameters whose settings are not masked by SHOW DATABASE...ALL, otherwise Vertica returns an error. If you specify a single parameter that is not set, SHOW DATABASE returns an empty row for that parameter. To obtain the names of database-level parameters, query system table \texttt{CONFIGURATION_PARAMETERS}.</td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>Shows all configuration parameters set at the database level. For non-superusers, Vertica masks settings of security parameters, which only superusers can access.</td>
</tr>
</tbody>
</table>

Privileges

- **Superuser**: Shows all database parameter settings.
- **Non-superuser**: Masks all security parameter settings, which only superusers can access. To determine which parameters require superuser privileges, query system table \texttt{CONFIGURATION_PARAMETERS}.

Examples

Show to a non-superuser all configuration parameters that are set on the database:

\[
=> \text{SHOW DATABASE DEFAULT ALL;}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
</table>

Vertica Analytic Database (9.0.x)
SHOW settings for two configuration parameters:

```sql
=> SHOW DATABASE DEFAULT AllowNumericOverflow, NumericSumExtraPrecisionDigits;
```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowNumericOverflow</td>
<td>1</td>
</tr>
<tr>
<td>NumericSumExtraPrecisionDigits</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

**SHOW NODE**

Displays configuration parameter values that are set for a node. If you specify a parameter that is not set, SHOW NODE returns an empty row for that parameter.

**Note:** If the configuration parameter is set but requires a database restart to take effect, the value shown might differ from the active value.

### Syntax

```sql
SHOW NODE node-name { parameter-name [, ...] | ALL }
```

### Parameters

<table>
<thead>
<tr>
<th><strong>node-name</strong></th>
<th>Name of the target node.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parameter-name</strong></td>
<td>Names of one or more node-level configuration parameters. To obtain the names of node-level parameters, query system table <code>CONFIGURATION_PARAMETERS</code>.</td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>Shows all configuration parameters set at the node level.</td>
</tr>
</tbody>
</table>
Privileges

None

Examples

View all configuration parameters and their settings for node v_vmart_node0001:

```sql
=> SHOW NODE v_vmart_node0001 ALL;

name          | setting
---------------|---------
DefaultIdleSessionTimeout | 5 hour  
MaxClientSessions     | 20
```

**SHOW SESSION**

Displays configuration parameter values that are set for the current session. If you specify a parameter that is not set, SHOW SESSION returns an empty row for that parameter.

Note: If the configuration parameter is set but requires a database restart to take effect, the value shown might differ from the active value.

Syntax

```sql
SHOW SESSION { parameter-name[,...] | ALL }
```

Parameters

<table>
<thead>
<tr>
<th>parameter-name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter-name</td>
<td>Names of one or more session-level configuration parameters. To obtain the names of session-level parameters, query system table <code>CONFIGURATION_PARAMETERS</code>.</td>
</tr>
<tr>
<td>ALL</td>
<td>Shows all configuration parameters set at the session level.</td>
</tr>
</tbody>
</table>
Privileges

None

Examples

View all configuration parameters and their settings for the current session:

```sql
=> SHOW SESSION ALL;
```

```

<table>
<thead>
<tr>
<th>name</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>en_US@collation=binary (LEN_KBINARY)</td>
</tr>
<tr>
<td>autocommit</td>
<td>off</td>
</tr>
<tr>
<td>standard_conforming_strings</td>
<td>on</td>
</tr>
<tr>
<td>escape_string_warning</td>
<td>on</td>
</tr>
<tr>
<td>datestyle</td>
<td>ISO, MDY</td>
</tr>
<tr>
<td>intervalstyle</td>
<td>plain</td>
</tr>
<tr>
<td>timezone</td>
<td>America/New_York</td>
</tr>
<tr>
<td>search_path</td>
<td>&quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
<tr>
<td>transaction_isolation</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>transaction_read_only</td>
<td>false</td>
</tr>
<tr>
<td>resource_pool</td>
<td>general</td>
</tr>
<tr>
<td>memorycap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>tempspacecap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>runtimecap</td>
<td>UNLIMITED</td>
</tr>
<tr>
<td>enabled roles</td>
<td>dbduser*, dbadmin*, pseudosuperuser*</td>
</tr>
<tr>
<td>available roles</td>
<td>dbduser*, dbadmin*, pseudosuperuser*</td>
</tr>
<tr>
<td>ForceUDxFencedMode</td>
<td>1</td>
</tr>
</tbody>
</table>

(17 rows)
```

**START TRANSACTION**

Starts a transaction block.

**Syntax**

```sql
START TRANSACTION [ isolation_level ]
```

where `isolation_level` is one of:

```sql
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }READ { ONLY | WRITE }
```
Parameters

Isolation level, described in the following table, determines what data the transaction can access when other transactions are running concurrently. The isolation level cannot be changed after the first query (SELECT) or DML statement (INSERT, DELETE, UPDATE) has run. A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

<table>
<thead>
<tr>
<th>WORK</th>
<th>TRANSACTION</th>
<th>Have no effect; they are optional keywords for readability.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOLATION LEVEL {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td></td>
<td>SERIALIZABLE—Sets the strictest level of SQL transaction isolation. This level emulates transactions serially, rather than concurrently. It holds locks and blocks write operations until the transaction completes. Not recommended for normal query operations.</td>
</tr>
<tr>
<td>REPEATABLE READ</td>
<td></td>
<td>REPEATABLE READ—Automatically converted to SERIALIZABLE by Vertica.</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td></td>
<td>READ COMMITTED (Default)—Allows concurrent transactions. Use READ COMMITTED isolation for normal query operations, but be aware that there is a subtle difference between them. See Transactions for more information.</td>
</tr>
<tr>
<td>READ UNCOMMITTED</td>
<td></td>
<td>READ UNCOMMITTED—Automatically converted to READ COMMITTED by Vertica.</td>
</tr>
<tr>
<td>READ {WRITE</td>
<td>ONLY}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSERT, UPDATE, DELETE, and COPY if the table they would write to is not a temporary table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All CREATE, ALTER, and DROP commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GRANT, REVOKE, and EXPLAIN if the command it would run is among those listed.</td>
</tr>
</tbody>
</table>
Privileges

No special permissions required.

Notes

BEGIN performs the same function as START TRANSACTION.

Examples

This example shows how to start a transaction.

```
= > START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
START TRANSACTION
=> CREATE TABLE sample_table (a INT);
CREATE TABLE
=> INSERT INTO sample_table (a) VALUES (1);
OUTPUT
--------
1
(1 row)
```

See Also

- Transactions
- Creating Transactions
- COMMIT
- END
- ROLLBACK
**TRUNCATE TABLE**

Removes all storage associated with a table, while leaving the table definition intact. TRUNCATE TABLE auto-commits the current transaction after statement execution and cannot be rolled back.

TRUNCATE TABLE removes all table history preceding the current epoch, regardless of where that data resides (WOS or ROS) or how it is segmented. Immediately after TRUNCATE TABLE returns, AT EPOCH queries on the truncated table return nothing.

**Syntax**

```
TRUNCATE TABLE [[database.]schema.]table-name
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[database.]schema</code></td>
<td>Specifies a schema, by default public. If <code>schema</code> is any schema other than public, you must supply the schema name. For example: <code>myschema.thisDbObject</code> If you specify a database, it must be the current database.</td>
</tr>
<tr>
<td><code>table-name</code></td>
<td>The name of the anchor table or temporary table to truncate. You cannot truncate an external table.</td>
</tr>
</tbody>
</table>

**Privileges**

One of the following privileges is required:

- Superuser
- Table owner
- User with USAGE privileges on the table's schema—see GRANT (Schema)

A schema owner can drop a table but cannot truncate a table.
Examples

See Truncating Tables in the Administrator's Guide.

See Also

- DELETE
- DROP TABLE
- Best Practices for DELETE and UPDATE

**UPDATE**

Replaces the values of the specified columns in all rows for which a specific condition is true. All other columns and rows in the table are unchanged. If successful, UPDATE returns the number of rows updated. A count of 0 indicates no rows matched the condition.

UPDATE inserts new records into the WOS and marks the old records for deletion. If the WOS fills up, the operation overflows to the ROS.

**Syntax**

```
... SET set-expression [, ... ]
... [ FROM from-list ]
... [ where-clause ]
```

**Parameters**

<table>
<thead>
<tr>
<th>/*+ hint [, hint] */</th>
<th>One or both of the following hints:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A load hint, one of the following: AUTO, DIRECT, or TRICKLE</td>
</tr>
<tr>
<td></td>
<td>LABEL</td>
</tr>
</tbody>
</table>
| **database.**schema | Specifies a schema, by default public. If **schema** is any schema other than public, you must supply the schema name. For example:
| | | `myschema.thisDbObject`
| If you specify a database, it must be the current database. |
| **table-reference** | Specifies a table, one of the following:
| | - An optionally qualified table name with optional table aliases, column aliases, and outer joins.
| | - An outer join table.
| You cannot update a projection. |
| **alias** | A temporary name used to reference the table. |
| **SET** | Specifies the columns to update. Each **set-expression** in the SET clause specifies a target column and its new value as follows:
| **set-expression** [,....] | `column-name = { expression | DEFAULT }`
| where:
| | - **column-name** is any column that does not have primary key or foreign key referential integrity constraints.
| | - **expression** specifies a value to assign to the column. The expression can use the current values of this and other table columns. For example:
| | | `UPDATE T1 SET C1 = C1+1`
| | UPDATE only modifies the columns specified by the SET clause. Unspecified columns remain unchanged. |
| **FROM from-list** | A list of table expressions, allowing columns from other tables to appear in the WHERE condition and the UPDATE expressions. This is similar to the list of tables that can be specified in the **FROM Clause** of a SELECT command.
| **Important:** from-list must not include the target table. |
Privileges

Table owner or user with GRANT OPTION is grantor.

- UPDATE privilege on table
- USAGE privilege on schema that contains the table
- SELECT privilege on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause

Subqueries and Joins

UPDATE supports subqueries and joins, which is useful for updating values in a table based on values that are stored in other tables. For details, see Subqueries in UPDATE and DELETE Statements in Analyzing Data.

Restrictions

- The table you specify in the UPDATE list cannot also appear in the FROM list (no self joins); for example, the following UPDATE statement is not allowed:

```sql
BEGIN;
UPDATE result_table
SET address='new' || r2.address
FROM result_table r2
WHERE r2.cust_id = result_table.cust_id + 10;
ERROR: Self joins in UPDATE statements are not allowed
DETAIL: Target relation result_table also appears in the FROM list
```

- If the joins specified in the WHERE predicate produce more than one copy of the row in the table to be updated, the new value of the row in the table is chosen arbitrarily.

- If any primary key, unique key, or check constraints are enabled for automatic enforcement, Vertica enforces those constraints when you insert values into a table. If a violation occurs, Vertica rolls back the SQL statement and returns an error. This behavior occurs for INSERT, UPDATE, COPY, and MERGE SQL statements.
Note: Automatic constraint enforcement requires that you have SELECT privileges on the table containing the constraint.

Examples

In the fact table, modify the price column value for all rows where the cost column value is greater than 100:

=> UPDATE fact SET price = price - cost * 80 WHERE cost > 100;

In the retail.customer table, set the state column to NH when the CID column value is greater than 100:

=> UPDATE retail.customer SET state = 'NH' WHERE CID > 100;

To use table aliases in UPDATE queries, consider the following two tables:

<table>
<thead>
<tr>
<th>cust_id</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Lincoln Street</td>
</tr>
<tr>
<td>30</td>
<td>Beach Avenue</td>
</tr>
<tr>
<td>30</td>
<td>Booth Hill Road</td>
</tr>
<tr>
<td>40</td>
<td>Mt. Vernon Street</td>
</tr>
<tr>
<td>50</td>
<td>Hillside Avenue</td>
</tr>
</tbody>
</table>

(5 rows)

<table>
<thead>
<tr>
<th>new_cust_id</th>
<th>new_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Infinite Loop</td>
</tr>
<tr>
<td>30</td>
<td>Loop Infinite</td>
</tr>
<tr>
<td>60</td>
<td>New Addresses</td>
</tr>
</tbody>
</table>

(3 rows)

The following query and subquery use table aliases to update the address column in result_table (alias r) with the new address from the corresponding column in the new_addresses table (alias n):

=> UPDATE result_table r
   SET address=n.new_address
   FROM new_addresses n
   WHERE r.cust_id = n.new_cust_id;

result_table shows the address field updates made for customer IDs 20 and 30:

<table>
<thead>
<tr>
<th>cust_id</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)
<table>
<thead>
<tr>
<th></th>
<th>Infinite Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Loop Infinite</td>
</tr>
<tr>
<td>30</td>
<td>Loop Infinite</td>
</tr>
<tr>
<td>40</td>
<td>Mt. Vernon Street</td>
</tr>
<tr>
<td>50</td>
<td>Hillside Avenue</td>
</tr>
</tbody>
</table>

(5 rows)
Vertica System Tables

Vertica provides system tables that let you monitor your database. Query these tables the same way you perform query operations on base or temporary tables—by using SELECT statements.

For more information, see the following sections in the Administrator's Guide:

- Using System Tables
- Monitoring Vertica
V_CATALOG Schema

The system tables in this section reside in the v_catalog schema. These tables provide information (metadata) about the objects in a database; for example, tables, constraints, users, projections, and so on.

ACCESS_POLICY

Provides information about access policies associated with specific tables.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_POLICY_OID</td>
<td>INTEGER</td>
<td>A unique identifier for the access policy.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the table with which the access policy is assigned.</td>
</tr>
<tr>
<td>IS_POLICY_ENABLED</td>
<td>BOOLEAN</td>
<td>Indicates whether or not you have enabled the access policy.</td>
</tr>
<tr>
<td>POLICY_TYPE</td>
<td>VARCHAR</td>
<td>The type of access policy assigned to the table, valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column Policy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Row Policy</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>VARCHAR</td>
<td>The expression used when creating the access policy.</td>
</tr>
<tr>
<td>TRIGGERED_BY</td>
<td>VARCHAR</td>
<td>The SQL statement that triggers the access policy.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The column to which the access policy is assigned.</td>
</tr>
</tbody>
</table>

ALL_TABLES

Provides summary information about tables in a Vertica database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema that contains the table.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name.</td>
</tr>
<tr>
<td>TABLE_TYPE</td>
<td>VARCHAR</td>
<td>The type of table, which can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SYSTEM TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VIEW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GLOBAL TEMPORARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOCAL TEMPORARY</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR</td>
<td>A brief comment about the table. You define this field by using the <code>COMMENT ON TABLE</code> and <code>COMMENT ON VIEW</code> commands.</td>
</tr>
</tbody>
</table>

**Example**

```sql
onenode=> SELECT DISTINCT table_name, table_type FROM all_tables
             WHERE table_name ILIKE 't%';
    table_name | table_type
--------------------------
types | SYSTEM TABLE
trades | TABLE
tuple_mover_operations | SYSTEM TABLE
tables | SYSTEM TABLE
tuning_recommendations | SYSTEM TABLE
testid | TABLE
table_constraints | SYSTEM TABLE
transactions | SYSTEM TABLE
(8 rows)
onenode=> SELECT table_name, table_type FROM all_tables
             WHERE table_name ILIKE 'my%';
   table_name | table_type
--------------------------
mystocks | VIEW
(1 row)
=> SELECT * FROM all_tables LIMIT 4;
[ RECORD 1 ]----------------------------------
schema_name | v_catalog
table_id | 10206
table_name | all_tables```
### CLIENT_AUTH

Provides information about the client authentication methods that you created.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_OID</td>
<td>INTEGER</td>
<td>Unique identifier for the authentication method.</td>
</tr>
<tr>
<td>AUTH_NAME</td>
<td>VARCHAR</td>
<td>Name that you gave to the authentication method.</td>
</tr>
<tr>
<td>IS_AUTH_ENABLED</td>
<td>BOOLEAN</td>
<td>Indicates whether you enabled the authentication method.</td>
</tr>
<tr>
<td>AUTH_HOST_TYPE</td>
<td>VARCHAR</td>
<td>The authentication host type, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- LOCAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- HOST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- HOSTSSL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- HOSTNOSSL</td>
</tr>
<tr>
<td>AUTH_HOST_ADDRESS</td>
<td>VARCHAR</td>
<td>If AUTH_HOST_TYPE is HOST, AUTH_HOST_ADDRESS is the IP address (or address range) of the remote host.</td>
</tr>
<tr>
<td>AUTH_METHOD</td>
<td>VARCHAR</td>
<td>Authentication method to be used.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>v_ident</td>
<td>True</td>
<td>LOCAL</td>
</tr>
<tr>
<td>v_gss</td>
<td>True</td>
<td>HOST</td>
</tr>
<tr>
<td>v_trust</td>
<td>False</td>
<td>LOCAL</td>
</tr>
<tr>
<td>v_ldap</td>
<td>True</td>
<td>HOST</td>
</tr>
<tr>
<td>RejectNoSSL</td>
<td>True</td>
<td>HOSTNOSSL</td>
</tr>
<tr>
<td>RejectWithSSL</td>
<td>True</td>
<td>HOSTSSL</td>
</tr>
<tr>
<td>v_hash</td>
<td>False</td>
<td>LOCAL</td>
</tr>
<tr>
<td>v_tls</td>
<td>True</td>
<td>HOSTSSL</td>
</tr>
<tr>
<td>v_trust</td>
<td>True</td>
<td>HOSTSSL</td>
</tr>
</tbody>
</table>

Examples

This example shows how to get information about each client authentication method that you created:

```sql
=> SELECT auth_name, is_auth_enabled, auth_host_type, auth_host_address, auth_method FROM CLIENT.AUTH;
```

<table>
<thead>
<tr>
<th>auth_name</th>
<th>is_auth_enabled</th>
<th>auth_host_type</th>
<th>auth_host_address</th>
<th>auth_method</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_ident</td>
<td>True</td>
<td>LOCAL</td>
<td></td>
<td>IDENT</td>
</tr>
<tr>
<td>v_gss</td>
<td>True</td>
<td>HOST</td>
<td>0.0.0.0/0</td>
<td>GSS</td>
</tr>
<tr>
<td>v_trust</td>
<td>False</td>
<td>LOCAL</td>
<td></td>
<td>TRUST</td>
</tr>
<tr>
<td>v_ldap</td>
<td>True</td>
<td>HOST</td>
<td>10.19.133.123/</td>
<td>LDAP</td>
</tr>
<tr>
<td>RejectNoSSL</td>
<td>True</td>
<td>HOSTNOSSL</td>
<td>0.0.0.0/0</td>
<td>REJECT</td>
</tr>
<tr>
<td>RejectWithSSL</td>
<td>True</td>
<td>HOSTSSL</td>
<td>0.0.0.0/0</td>
<td>REJECT</td>
</tr>
<tr>
<td>v_hash</td>
<td>False</td>
<td>LOCAL</td>
<td></td>
<td>HASH</td>
</tr>
<tr>
<td>v_tls</td>
<td>True</td>
<td>HOSTSSL</td>
<td>1.1.1.1/0</td>
<td>TLS</td>
</tr>
<tr>
<td>v_trust</td>
<td>True</td>
<td>HOSTSSL</td>
<td>2001:db8:ab:123/128</td>
<td>TLS</td>
</tr>
</tbody>
</table>

9 rows

CLIENT_AUTH_PARAMS

Provides information about client authentication methods that have parameter values assigned.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_OID</td>
<td>INTEGER</td>
<td>A unique identifier for the authentication method.</td>
</tr>
<tr>
<td>AUTH_NAME</td>
<td>VARCHAR</td>
<td>Name that you defined for the authentication method.</td>
</tr>
<tr>
<td>AUTH_PARAMETER_NAME</td>
<td>VARCHAR</td>
<td>Parameter name required by the authentication method. Some examples are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- system_users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- binddn_prefix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- host</td>
</tr>
<tr>
<td>AUTH_PARAMETER_VALUE</td>
<td>VARCHAR</td>
<td>Value of the specified parameter.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows how to retrieve parameter names and values for all authentication methods that you created. The authentication methods that have parameters are:

- **v_ident**
- **v_ldap**
- **v_ldap1**

```sql
=> SELECT * FROM CLIENT_AUTH_PARAMS;

<table>
<thead>
<tr>
<th>auth_oid</th>
<th>auth_name</th>
<th>auth_parameter_name</th>
<th>auth_parameter_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273741304</td>
<td>v_ident</td>
<td>system_users</td>
<td>root</td>
</tr>
<tr>
<td>45035996273741332</td>
<td>v_gss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273741350</td>
<td>v_password</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273741368</td>
<td>v_trust</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273741388</td>
<td>v_ldap</td>
<td>host</td>
<td>ldap://172.16.65.177</td>
</tr>
<tr>
<td>45035996273741388</td>
<td>v_ldap</td>
<td>binddn_prefix</td>
<td>cn=</td>
</tr>
<tr>
<td>45035996273741388</td>
<td>v_ldap</td>
<td>binddn_suffix</td>
<td>,dc=qa_domain,dc=com</td>
</tr>
<tr>
<td>45035996273741406</td>
<td>RejectNoSSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273741424</td>
<td>RejectWithSSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273741450</td>
<td>v_md5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273904044</td>
<td>l_tls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273906566</td>
<td>v_hash</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45035996273918432</td>
<td>v_ldap1</td>
<td>host</td>
<td>ldap://172.16.65.177</td>
</tr>
<tr>
<td>45035996273918432</td>
<td>v_ldap1</td>
<td>basedn</td>
<td>dc=qa_domain,dc=com</td>
</tr>
<tr>
<td>45035996273918432</td>
<td>v_ldap1</td>
<td>binddn</td>
<td>cn=Manager,dc=qa_domain,dc=com</td>
</tr>
<tr>
<td>45035996273918432</td>
<td>v_ldap1</td>
<td>bind_password</td>
<td>secret</td>
</tr>
<tr>
<td>45035996273918432</td>
<td>v_ldap1</td>
<td>search_attribute</td>
<td>cn</td>
</tr>
</tbody>
</table>

(17 rows)
CLUSTER_LAYOUT

Shows the relative position of the actual arrangement of the nodes participating in the cluster and the fault groups that affect them. Ephemeral nodes are not shown in the cluster layout ring because they hold no resident data.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTER_POSITION</td>
<td>INTEGER</td>
<td>Position of the node in the cluster ring, counting forward from 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> An output value of 0 has no special meaning other than there are no nodes in position before the node assigned 0.</td>
</tr>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the node.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node in the cluster ring. Only permanent nodes participating in database activity appear in the cluster layout. Ephemeral nodes are not shown in the output.</td>
</tr>
<tr>
<td>FAULT_GROUP_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the fault group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This value matches the FAULT_GROUP.MEMBER_ID value, but only if this node is in a fault group; otherwise the value is NULL.</td>
</tr>
<tr>
<td>FAULT_GROUP_NAME</td>
<td>VARCHAR</td>
<td>The name of the fault group for the node.</td>
</tr>
<tr>
<td>FAULT_GROUP_TIER</td>
<td>INTEGER</td>
<td>The node's depth in the fault group tree hierarchy. For example is the node:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Is not in a fault group, output is null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Is in the top level fault group, output is 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Is in a fault group's child, output is 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Is a fault group's grandchild, output is 2</td>
</tr>
</tbody>
</table>
### COLUMNS

Provides table column information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema name for which information is listed in the database.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Table name for which information is listed in the database.</td>
</tr>
<tr>
<td>IS_SYSTEM_TABLE</td>
<td>BOOLEAN</td>
<td>Specifies whether the table is a system table.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>VARCHAR</td>
<td>A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The column name for which information is listed in the database.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR</td>
<td>Column's data type, for example VARCHAR(16), INTEGER, or FLOAT.</td>
</tr>
<tr>
<td>DATA_TYPE_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the data type.</td>
</tr>
<tr>
<td>DATA_TYPE_LENGTH</td>
<td>INTEGER</td>
<td>Maximum allowable length of the data type.</td>
</tr>
<tr>
<td>CHARACTER_MAXIMUM_LENGTH</td>
<td>VARCHAR</td>
<td>Maximum allowable length of the column.</td>
</tr>
<tr>
<td>NUMERIC_PRECISION</td>
<td>INTEGER</td>
<td>Number of significant decimal digits.</td>
</tr>
<tr>
<td>NUMERIC_SCALE</td>
<td>INTEGER</td>
<td>Number of fractional digits.</td>
</tr>
<tr>
<td>DATETIME_PRECISION</td>
<td>INTEGER</td>
<td>For TIMESTMP data type, returns the declared precision; returns NULL if no precision was declared.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INTERVAL_PRECISION</td>
<td>INTEGER</td>
<td>Number of fractional digits retained in the seconds field.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>INTEGER</td>
<td>Column position respective to other columns in the table.</td>
</tr>
<tr>
<td>IS_NULLABLE</td>
<td>BOOLEAN</td>
<td>Specifies whether the column can contain NULL values.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT</td>
<td>VARCHAR</td>
<td>Expression set on a column with the constraint DEFAULT.</td>
</tr>
<tr>
<td>COLUMN_SET_USING</td>
<td>VARCHAR</td>
<td>Expression set on a column with the constraint SET USING.</td>
</tr>
<tr>
<td>IS_IDENTITY</td>
<td>BOOLEAN</td>
<td>Specifies whether the column is an identity column. See Column-Constraint.</td>
</tr>
</tbody>
</table>

### Examples

Retrieve table and column information from the COLUMNS table:

```sql
=> SELECT table_schema, table_name, column_name, data_type, is_nullable
    FROM columns WHERE table_schema = 'store'
    AND data_type = 'Date';
```

<table>
<thead>
<tr>
<th>table_schema</th>
<th>table_name</th>
<th>column_name</th>
<th>data_type</th>
<th>is_nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>first_open_date</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>last_remodel_date</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td>store</td>
<td>store_orders_fact</td>
<td>date_ordered</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td>store</td>
<td>store_orders_fact</td>
<td>date_shipped</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td>store</td>
<td>store_orders_fact</td>
<td>expected_delivery_date</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td>store</td>
<td>store_orders_fact</td>
<td>date_delivered</td>
<td>Date</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6 rows)

DATETIME_PRECISION is NULL because the table definition declares no precision:

```sql
=> CREATE TABLE c (c TIMESTAMP);
CREATE TABLE c
=> SELECT table_name, column_name, datetime_precision FROM columns
    WHERE table_name = 'c';
```

<table>
<thead>
<tr>
<th>table_name</th>
<th>column_name</th>
<th>datetime_precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

1 row)

DATETIME_PRECISION is 4 because the table definition declares precision as 4:
An identity column is a sequence available only for numeric column types. To identify what column in a table, if any, is an identity column, search the COLUMNS table to find the identity column in a table testid:

```
=> CREATE TABLE testid (c1 IDENTITY(1, 1, 1000), c2 INT);
=> \x
Expanded display is on.
=> SELECT * FROM COLUMNS WHERE is_identity='t' AND table_name='testid';
- RECORD 1 ]------------------------
table_id | 45035996273719486
table_schema | public
table_name | testid
is_system_table | f
column_id | 45035996273719486-1
column_name | c1
data_type | int
data_type_id | 6
data_type_length | 8
character_maximum_length | 
text | 
numeric_precision | 
numeric_scale | 
datetime_precision | 
interval_precision | 
ordinal_position | 1
is_nullable | f
column_default | 
is_identity | t
```

Use the SEQUENCES table to get detailed information about the sequence in testid:

```
=> SELECT * FROM sequences WHERE identity_table_name='testid';
- RECORD 1 ]------------------------
sequence_schema | public
sequence_name | testid_c1_seq
owner_name | dbadmin
identity_table_name | testid
session_cache_count | 1000
allow_cycle | f
output_ordered | f
increment_by | 1
minimum | 1
maximum | 9223372036854775807
current_value | 0
sequence_schema_id | 45035996273704976
sequence_id | 45035996273719488
owner_id | 45035996273704962
identity_table_id | 45035996273719486
```

For more information about sequences and identity columns, see Sequence Types.
COMMENTS

Returns information about comments associated with objects in the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT_ID</td>
<td>INTEGER</td>
<td>The comment's internal ID number</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>INTEGER</td>
<td>The internal ID number of the object associated with the comment</td>
</tr>
<tr>
<td>OBJECT_TYPE</td>
<td>VARCHAR</td>
<td>The type of object associated with the comment. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COLUMN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- CONSTRAINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FUNCTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- LIBRARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- NODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PROJECTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SCHEMA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SEQUENCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- VIEW</td>
</tr>
<tr>
<td>OBJECT_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema containing the object.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>The name of the object associated with the comment.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>VARCHAR</td>
<td>The internal ID of the owner of the object.</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR</td>
<td>The object owner's name.</td>
</tr>
<tr>
<td>CREATION_TIME</td>
<td>TIMESTAMPTZ</td>
<td>When the comment was created.</td>
</tr>
</tbody>
</table>
### LAST_MODIFIED_TIME
- **Data Type**: TIMESTAMPTZ
- **Description**: When the comment was last modified.

### COMMENT
- **Data Type**: VARCHAR
- **Description**: The text of the comments.

## CONSTRAINT_COLUMNS
Records information about table column constraints.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the schema that contains this table.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the table in which the column resides.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>Name of the column that is constrained. For check constraints, if more than one column is referenced, each appears as a separate row.</td>
</tr>
<tr>
<td>CONSTRAINT_NAME</td>
<td>VARCHAR</td>
<td>Constraint name for which information is listed.</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>CHAR</td>
<td>Indicates the constraint type. Valid Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- c — check</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- f — foreign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- n — not null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- p — primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- u — unique</td>
</tr>
<tr>
<td>IS_ENABLED</td>
<td>BOOLEAN</td>
<td>Indicates if a constraint for a primary key, unique key, or check constraint is currently enabled. Can be t (True) or f (False).</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
REFERENCE_TABLE_ID | INTEGER | A unique numeric ID assigned by the Vertica catalog that identifies the referenced table.
REFERENCE_TABLE_SCHEMA | VARCHAR | Schema name for which information is listed.
REFERENCE_TABLE_NAME | VARCHAR | References the TABLE_NAME column in the PRIMARY_KEY table.
REFERENCE_COLUMN_NAME | VARCHAR | References the COLUMN_NAME column in the PRIMARY_KEY table.

### Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

### DATABASES

Provides information about the databases in this Vertica installation.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_ID</td>
<td>INTEGER</td>
<td>The database's internal ID number</td>
</tr>
<tr>
<td>DATABASE_NAME</td>
<td>VARCHAR</td>
<td>The database's name</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>The database owner's ID</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>INTEGER</td>
<td>The database owner's name</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMPTZ</td>
<td>The date and time the database last started</td>
</tr>
<tr>
<td>COMPLIANCE_MESSAGE</td>
<td>VARCHAR</td>
<td>Message describing the current state of the database's license compliance.</td>
</tr>
<tr>
<td>EXPORT_SUBNET</td>
<td>VARCHAR</td>
<td>Can be either of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The subnet (on the public network) used by the database for import/export.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The public address of the subnet that the</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
| | | Vertica native load balancing uses for load balancing.
| LOAD_BALANCE_POLICY | VARCHAR | The current native connection load balance policy, which controls whether client connection requests are redirected to other hosts in the database. See [About Native Connection Load Balancing](#) in the Administrator's Guide.

#### Example

```sql
=> SELECT * FROM DATABASES;
```

<table>
<thead>
<tr>
<th>database_id</th>
<th>database_name</th>
<th>owner_id</th>
<th>owner_name</th>
<th>start_time</th>
<th>compliance_message</th>
<th>export_subnet</th>
<th>load_balance_policy</th>
<th>backend_address_family</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273704976</td>
<td>VMart</td>
<td>45035996273704962</td>
<td>dbadmin</td>
<td>2017-10-22 05:16:22.066961-04</td>
<td>The database is in compliance with respect to raw data size.</td>
<td>0</td>
<td>none</td>
<td>ipv4</td>
</tr>
</tbody>
</table>

**DIRECTED_QUERIES**

Returns information about directed queries.

| Column Name | Data Type | Description
---|---|---
| query_name | VARCHAR | This directed query's unique identifier, used by statements such as [ACTIVATE DIRECTED QUERY](#).
| is_active | BOOLEAN | Specifies whether the directed query is active.
| vertica_version | VARCHAR | The Vertica version used when this directed query was created.
| comment | VARCHAR | A user-supplied comment specified on creation of the directed query, up to 128 characters.
| creation_ | TIMESTAMPTZ | Specifies when the directed query was created.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input_query</td>
<td>VARCHAR</td>
<td>The input query that is associated with this directed query. Multiple directed queries can map to the same input query.</td>
</tr>
<tr>
<td>annotated_query</td>
<td>VARCHAR</td>
<td>The directed query that was saved with <code>CREATE DIRECTED QUERY</code>.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**Truncated Query Results**

Query results for the fields `input_query` and `annotated_query` are truncated after 8192 characters. You can get the full content of both fields in two ways:

- Use the statement `GET DIRECTED QUERY`.
- Use `EXPORT_CATALOG` to export directed queries.

**DUAL**

DUAL is a single-column "dummy" table with one record whose value is X; for example:

```sql
=> SELECT * FROM DUAL;
  dummy
  ------
  X
(1 row)
```

You can write the following types of queries:

```sql
=> SELECT 1 FROM dual;
?column?
-------
1
(1 row)
=> SELECT current_timestamp, current_user FROM dual;
?column? | current_user
--------- |----------
```
Restrictions

You cannot create projections for DUAL.

ELASTIC_CLUSTER

Returns information about cluster elasticity, such as whether Elastic Cluster is running.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCALING_FACTOR</td>
<td>INTEGER</td>
<td>This value is only meaningful when you enable local segments. SCALING_FACTOR influences the number of local segments on each node. Initially—before a rebalance runs—there are scaling_factor number of local segments per node. A large SCALING_FACTOR is good for rebalancing a potentially wide range of cluster configurations quickly. However, too large a value could lead to ROS pushback, particularly in a database with a table with a large number of partitions. See SET_SCALING_FACTOR for more details.</td>
</tr>
<tr>
<td>MAXIMUM_SKEW_PERCENT</td>
<td>INTEGER</td>
<td>This value is only meaningful when you enable local segments. MAXIMUM_SKEW_PERCENT is the maximum amount of skew a rebalance operation tolerates, which preferentially redistributes local segments; however, if after doing so the segment ranges of any two nodes differs by more than this amount, rebalance will separate and distribute storage to even the distribution.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SEGMENT_LAYOUT</td>
<td>VARCHAR</td>
<td>Current, offset=0, segment layout. New segmented projections will be created with this layout, with segments rotated by the corresponding offset. Existing segmented projections will be rebalanced into an offset of this layout.</td>
</tr>
<tr>
<td>LOCAL_SEGMENT_LAYOUT</td>
<td>VARCHAR</td>
<td>Similar to SEGMENT_LAYOUT but includes details that indicate the number of local segments, their relative size and node assignment.</td>
</tr>
<tr>
<td>VERSION</td>
<td>INTEGER</td>
<td>Number that gets incremented each time the cluster topology changes (nodes added, marked ephemeral, marked permanent, etc). Useful for monitoring active and past rebalance operations.</td>
</tr>
<tr>
<td>IS_ENABLED</td>
<td>BOOLEAN</td>
<td>True if Elastic Cluster is enabled, otherwise false.</td>
</tr>
<tr>
<td>IS_LOCAL_SEGMENT_ENABLED</td>
<td>BOOLEAN</td>
<td>True if local segments are enabled, otherwise false.</td>
</tr>
<tr>
<td>IS_REBALANCE_RUNNING</td>
<td>BOOLEAN</td>
<td>True if rebalance is currently running, otherwise false.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**See Also**

- ENABLE_ELASTIC_CLUSTER
- DISABLE_ELASTIC_CLUSTER
- Elastic Cluster

**EPOCHS**

For all epochs, provides the date and time of the close and the corresponding epoch number of the closed epoch. This information lets you determine which time periods pertain to which
epochs.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPOCH_CLOSE_TIME</td>
<td>DATETIME</td>
<td>The date and time of the close of the epoch.</td>
</tr>
<tr>
<td>EPOCH_NUMBER</td>
<td>INTEGER</td>
<td>The corresponding epoch number of the closed epoch.</td>
</tr>
</tbody>
</table>

See Also

- Epoch Management Parameters
- Epoch Management Functions

FAULT_GROUPS

View the fault groups and their hierarchy in the cluster.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the fault group.</td>
</tr>
<tr>
<td>MEMBER_TYPE</td>
<td>VARCHAR</td>
<td>The type of fault group. Values can be either NODE or FAULT GROUP.</td>
</tr>
<tr>
<td>MEMBER_NAME</td>
<td>VARCHAR</td>
<td>Name associated with this fault group. Values will be the node name or the fault group name.</td>
</tr>
<tr>
<td>PARENT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the parent fault group. The parent fault group can contain:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Other fault groups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Nodes and other fault groups</td>
</tr>
<tr>
<td>PARENT_TYPE</td>
<td>VARCHAR</td>
<td>The type of parent fault group, where the default/root parent is the DATABASE object. Can be one of the following objects:</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
| | | • FAULT GROUP  
| | | • DATABASE  

**PARENT_NAME** | VARCHAR | The name of the fault group that contains nodes or other fault groups or both nodes and fault groups.  
**IS AUTOMATICALLY GENERATED** | BOOLEAN | If true, denotes whether Vertica Analytic Database created fault groups for you to manage the fault tolerance of control nodes in large cluster configurations. If false, denotes that you created fault groups manually. See [Fault Groups](#) for more information

### Examples

**Show the current hierarchy of fault groups in the cluster:**

```sql
vmartdb=> SELECT member_type, member_name, parent_type, CASE WHEN parent_type = 'DATABASE' THEN '' ELSE parent_name END FROM fault_groups ORDER BY member_name;
```

<table>
<thead>
<tr>
<th>member_type</th>
<th>member_name</th>
<th>parent_type</th>
<th>parent_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE</td>
<td>v_vmart_node0001</td>
<td>FAULT GROUP</td>
<td>two</td>
</tr>
<tr>
<td>NODE</td>
<td>v_vmart_node0002</td>
<td>FAULT GROUP</td>
<td>two</td>
</tr>
<tr>
<td>NODE</td>
<td>v_vmart_node0003</td>
<td>FAULT GROUP</td>
<td>three</td>
</tr>
<tr>
<td>FAULT GROUP</td>
<td>one</td>
<td>DATABASE</td>
<td></td>
</tr>
<tr>
<td>FAULT GROUP</td>
<td>three</td>
<td>DATABASE</td>
<td></td>
</tr>
<tr>
<td>FAULT GROUP</td>
<td>two</td>
<td>FAULT GROUP</td>
<td>one</td>
</tr>
</tbody>
</table>

**View the distribution of the segment layout:**

```sql
vmartdb=> SELECT segment_layout from elastic_cluster;
```

```
 v_vmart_node0001[33.3%]  v_vmart_node0003[33.3%]  v_vmart_node0004[33.3%]
(1 row)
```
## See Also

- High Availability With Fault Groups in Vertica Concepts
- Fault Groups in the Administrator's Guide

### FOREIGN_KEYS

Provides foreign key information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.</td>
</tr>
<tr>
<td>CONSTRAINT_NAME</td>
<td>VARCHAR</td>
<td>The constraint name for which information is listed.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The name of the column that is constrained.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>VARCHAR</td>
<td>The position of the column within the key. The numbering of columns starts at 1.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name for which information is listed.</td>
</tr>
<tr>
<td>REFERENCE_TABLE_NAME</td>
<td>VARCHAR</td>
<td>References the TABLE_NAME column in the PRIMARY_KEY table.</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>VARCHAR</td>
<td>The constraint type, f, for foreign key.</td>
</tr>
<tr>
<td>REFERENCE_COLUMN_NAME</td>
<td>VARCHAR</td>
<td>References the COLUMN_NAME column in the PRIMARY_KEY table.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td>REFERENCE_TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>References the TABLE_SCHEMA column in the PRIMARY_KEY table.</td>
</tr>
</tbody>
</table>
Example

mydb=> SELECT constraint_name, table_name, ordinal_position, reference_table_name FROM foreign_keys ORDER BY 3;

<table>
<thead>
<tr>
<th>constraint_name</th>
<th>table_name</th>
<th>ordinal_position</th>
<th>reference_table_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>fk_store_sales_date</td>
<td>store_sales_fact</td>
<td>1</td>
<td>date_dimension</td>
</tr>
<tr>
<td>fk_online_sales_soldate</td>
<td>online_sales_fact</td>
<td>1</td>
<td>date_dimension</td>
</tr>
<tr>
<td>fk_store_orders_product</td>
<td>store_orders_fact</td>
<td>1</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_inventory_date</td>
<td>inventory_fact</td>
<td>1</td>
<td>date_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>inventory_fact</td>
<td>2</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_store_sales_product</td>
<td>store_sales_fact</td>
<td>2</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_store_orders_product</td>
<td>store_orders_fact</td>
<td>2</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_inventory_shipdate</td>
<td>online_sales_fact</td>
<td>3</td>
<td>store_dimension</td>
</tr>
<tr>
<td>fk_store_orders_product</td>
<td>store_orders_fact</td>
<td>3</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>inventory_fact</td>
<td>3</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>online_sales_fact</td>
<td>4</td>
<td>product_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>online_sales_fact</td>
<td>4</td>
<td>warehouse_dimension</td>
</tr>
<tr>
<td>fk_store_orders_vendor</td>
<td>store_orders_fact</td>
<td>4</td>
<td>vendor_dimension</td>
</tr>
<tr>
<td>fk_store_sales_store</td>
<td>store_sales_fact</td>
<td>5</td>
<td>store_dimension</td>
</tr>
<tr>
<td>fk_store_orders_employee</td>
<td>store_orders_fact</td>
<td>5</td>
<td>employee_dimension</td>
</tr>
<tr>
<td>fk_store_sales_promotion</td>
<td>store_sales_fact</td>
<td>5</td>
<td>promotion_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>online_sales_fact</td>
<td>6</td>
<td>customer_dimension</td>
</tr>
<tr>
<td>fk_store_sales_customer</td>
<td>online_sales_fact</td>
<td>6</td>
<td>customer_dimension</td>
</tr>
<tr>
<td>fk_inventory_product</td>
<td>online_sales_fact</td>
<td>7</td>
<td>call_center_dimension</td>
</tr>
<tr>
<td>fk_store_sales_employee</td>
<td>store_sales_fact</td>
<td>7</td>
<td>employee_dimension</td>
</tr>
<tr>
<td>fk_store_sales_op</td>
<td>online_sales_fact</td>
<td>8</td>
<td>online_page_dimension</td>
</tr>
<tr>
<td>fk_store_sales_shipping</td>
<td>online_sales_fact</td>
<td>8</td>
<td>shipping_dimension</td>
</tr>
<tr>
<td>fk_store_sales_warehouse</td>
<td>online_sales_fact</td>
<td>9</td>
<td>warehouse_dimension</td>
</tr>
<tr>
<td>fk_store_sales_promotion</td>
<td>online_sales_fact</td>
<td>10</td>
<td>promotion_dimension</td>
</tr>
</tbody>
</table>

(26 rows)

GRANTS

Provides information about privileges granted on various objects, the granting user, and grantee user. The order of columns in the table corresponds to the order in which they appear in the GRANT command. The GRANTS table does not retain the role grantor.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>The user being granted permission.</td>
</tr>
<tr>
<td>GRANTEE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRANT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the user granted permissions.</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>The user granting the permission.</td>
</tr>
<tr>
<td>GRANTOR_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the user who performed the grant operation.</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the object granted.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>The name of the object that is being granted privileges. Note that for schema privileges, the schemaname appears in the OBJECT_NAME column instead of the OBJECT_SCHEMA column.</td>
</tr>
<tr>
<td>OBJECT_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema that is being granted privileges.</td>
</tr>
<tr>
<td>OBJECT_TYPE</td>
<td>VARCHAR</td>
<td>The object type on which the grant was applied; for example, ROLE, SCHEMA, DATABASE, RESOURCEPOOL. Output from this column is useful in cases where a schema, resource pool, or user share the same name.</td>
</tr>
<tr>
<td>PRIVILEGES_DESCRIPTION</td>
<td>VARCHAR</td>
<td>A readable description of the privileges being granted; for example INSERT, SELECT. An asterisk in PRIVILEGES_DESCRIPTION output indicates a privilege WITH GRANT OPTION.</td>
</tr>
</tbody>
</table>

**Notes**

The `vsql` commands `\dp` and `\z` both include the schema name in the output. For example:

```sql
=> \dp
Access privileges for database "vmartdb"
Grantee | Grantor | Privileges | Schema | Name          
---------|---------|------------|--------|---------------
dbadmin | USAGE   |            | public          ```
The `vsql` command `\dp *.tablename` displays table names in all schemas. This command lets you distinguish grants for same-named tables in different schemas:

```
=> \dp *.events

<table>
<thead>
<tr>
<th>Grantee</th>
<th>Grantor</th>
<th>Privileges</th>
<th>Schema</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>user2</td>
<td>dbadmin</td>
<td>INSERT, SELECT, UPDATE, DELETE, REFERENCES</td>
<td>schema1</td>
<td>events</td>
</tr>
<tr>
<td>user1</td>
<td>dbadmin</td>
<td>SELECT</td>
<td>schema1</td>
<td>events</td>
</tr>
<tr>
<td>user2</td>
<td>dbadmin</td>
<td>INSERT, SELECT, UPDATE, DELETE, REFERENCES</td>
<td>schema2</td>
<td>events</td>
</tr>
<tr>
<td>user1</td>
<td>dbadmin</td>
<td>INSERT, SELECT</td>
<td>schema2</td>
<td>events</td>
</tr>
</tbody>
</table>
```

(4 rows)

The `vsql` command `\dp schemaname.*` displays all tables in the named schema:

```
=> \dp schema1.*

<table>
<thead>
<tr>
<th>Grantee</th>
<th>Grantor</th>
<th>Privileges</th>
<th>Schema</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>user2</td>
<td>dbadmin</td>
<td>INSERT, SELECT, UPDATE, DELETE, REFERENCES</td>
<td>schema1</td>
<td>events</td>
</tr>
<tr>
<td>user1</td>
<td>dbadmin</td>
<td>SELECT</td>
<td>schema1</td>
<td>events</td>
</tr>
</tbody>
</table>
```

(2 rows)

### Examples

This example shows `CREATE` and `USAGE` privileges granted to Bob in the fictitious apps database:

```
=> SELECT grantor, privileges_description, object_schema, object_name, grantee
FROM grants;

<table>
<thead>
<tr>
<th>grantor</th>
<th>privileges_description</th>
<th>object_schema</th>
<th>object_name</th>
<th>grantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbadmin</td>
<td>USAGE</td>
<td></td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>dbadmin</td>
<td>CREATE</td>
<td></td>
<td></td>
<td>Bob</td>
</tr>
</tbody>
</table>
```

(2 rows)

This next query looks for privileges granted to a particular set of grantees. The asterisk in privileges_description column for User1 means that user has WITH GRANT OPTION privileges.

```
=> SELECT grantor, privileges_description, object_schema, object_name, grantee
FROM grants WHERE grantee ILIKE 'User%';
```
In the following example, `online_sales` is the schema that first gets privileges, and then inside that schema the anchor table gets SELECT privileges:

```sql
=> SELECT grantee, grantor, privileges_description, object_schema, object_name
    FROM grants
    WHERE grantee='u1' ORDER BY object_name;
```

The following statement shows all grants for user Bob:

```sql
=> SELECT * FROM grants WHERE grantee = 'Bob';
```

<table>
<thead>
<tr>
<th>grantor</th>
<th>privileges_description</th>
<th>object_schema</th>
<th>object_name</th>
<th>grantees</th>
</tr>
</thead>
<tbody>
<tr>
<td>release</td>
<td>USAGE</td>
<td>general</td>
<td>User1</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>USAGE</td>
<td>general</td>
<td>User2</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>USAGE</td>
<td>general</td>
<td>User3</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>USAGE</td>
<td>s1</td>
<td>User1</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>USAGE</td>
<td>s1</td>
<td>User2</td>
<td></td>
</tr>
<tr>
<td>release</td>
<td>USAGE</td>
<td>s1</td>
<td>User3</td>
<td></td>
</tr>
<tr>
<td>User1</td>
<td>INSERT*, SELECT*, UPDATE*</td>
<td>s1</td>
<td>t1</td>
<td>User1</td>
</tr>
</tbody>
</table>

7 rows

```sql
(7 rows)
```
See Also

- HAS_ROLE
- ROLES
- USERS
- Managing Users and Privileges

HCATALOG_COLUMNS

Describes the columns of all tables available through the HCatalog Connector. Each row in this table corresponds to a column in a table accessible through the HCatalog Connector. See Using the HCatalog Connector in Integrating with Apache Hadoop for more information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR (128)</td>
<td>The name of the Vertica Analytic Database schema that contains the table containing this column</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HCATALOG_SCHEMA</td>
<td>VARCHAR (128)</td>
<td>The name of the Hive schema or database that contains the table containing this column</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR (128)</td>
<td>The name of the table that contains the column</td>
</tr>
<tr>
<td>IS_PARTITION_COLUMN</td>
<td>BOOLEAN</td>
<td>Whether the table is partitioned on this column</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR (128)</td>
<td>The name of the column</td>
</tr>
<tr>
<td>HCATALOG_DATA_TYPE</td>
<td>VARCHAR (128)</td>
<td>The Hive data type of this column</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR (128)</td>
<td>The Vertica Analytic Database data type of this column</td>
</tr>
<tr>
<td>DATA_TYPE_ID</td>
<td>INTEGER</td>
<td>Numeric ID of the column's Vertica Analytic Database data type</td>
</tr>
<tr>
<td>DATA_TYPE_LENGTH</td>
<td>INTEGER</td>
<td>The number of bytes used to store this data type</td>
</tr>
<tr>
<td>CHARACTER_MAXIMUM_LENGTH</td>
<td>INTEGER</td>
<td>For string data types, the maximum number of characters it can hold</td>
</tr>
<tr>
<td>NUMERIC_PRECISION</td>
<td>INTEGER</td>
<td>For numeric types, the precision of the values in the column</td>
</tr>
<tr>
<td>NUMERIC_SCALE</td>
<td>INTEGER</td>
<td>For numeric data types, the scale of the values in the column</td>
</tr>
<tr>
<td>DATETIME_PRECISION</td>
<td>INTEGER</td>
<td>For datetime data types, the precision of the values in the column</td>
</tr>
<tr>
<td>INTERVAL_PRECISION</td>
<td>INTEGER</td>
<td>For interval data types, the precision of the values in the column</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>INTEGER</td>
<td>The position of the column within the table</td>
</tr>
</tbody>
</table>
Privileges

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

Notes

If you are using WebHCat instead of HiveServer2, querying this table results in one web service call to the WebHCat server for each table in each HCatalog schema. If you need to perform multiple queries on this table in a short period of time, consider creating a copy of the table using a CREATE TABLE AS statement to improve performance. The copy does not reflect any changes made to the schema of the Hive tables after it was created, but it is much faster to query.

Example

The following example demonstrates finding the column information for a specific table:

```sql
=> SELECT * FROM HCATALOG_COLUMNS WHERE table_name = 'hcatalogtypes' -> ORDER BY ordinal_position;
-[ RECORD 1 ]----------------------------------------
table_schema | hcat
hcatalog_schema | default
table_name | hcatalogtypes
is_partition_column | f
column_name | intcol
hcatalog_data_type | int
data_type | int
data_type_id | 6
data_type_length | 8
character_maximum_length |
numeric_precision |
numeric_scale |
datetime_precision |
interval_precision |
ordinal_position | 1
-[ RECORD 2 ]----------------------------------------
table_schema | hcat
hcatalog_schema | default
table_name | hcatalogtypes
is_partition_column | f
column_name | floatcol
hcatalog_data_type | float
data_type | float
data_type_id | 7
data_type_length | 8
character_maximum_length |
```
<table>
<thead>
<tr>
<th>Numeric precision</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric scale</td>
<td></td>
</tr>
<tr>
<td>Datetime precision</td>
<td></td>
</tr>
<tr>
<td>Interval precision</td>
<td></td>
</tr>
<tr>
<td>Ordinal position</td>
<td>2</td>
</tr>
</tbody>
</table>

- [ RECORD 3 ]---------------------------------------------------
| Table schema       | hcat |
| Hcatalog schema    | default |
| Table name         | hcatalogtypes |
| Is partition column| f |
| Column name        | doublecol |
| Hcatalog data type | double |
| Data type          | float |
| Data type id       | 7 |
| Data type length   | 8 |
| Character maximum length |  |
| Numeric precision  |  |
| Numeric scale      |  |
| Datetime precision |  |
| Interval precision |  |
| Ordinal position   | 3 |

- [ RECORD 4 ]---------------------------------------------------
| Table schema       | hcat |
| Hcatalog schema    | default |
| Table name         | hcatalogtypes |
| Is partition column| f |
| Column name        | charcol |
| Hcatalog data type | string |
| Data type          | varchar(65000) |
| Data type id       | 9 |
| Data type length   | 65000 |
| Character maximum length | 65000 |
| Numeric precision  |  |
| Numeric scale      |  |
| Datetime precision |  |
| Interval precision |  |
| Ordinal position   | 4 |

- [ RECORD 5 ]---------------------------------------------------
| Table schema       | hcat |
| Hcatalog schema    | default |
| Table name         | hcatalogtypes |
| Is partition column| f |
| Column name        | varcharcol |
| Hcatalog data type | string |
| Data type          | varchar(65000) |
| Data type id       | 9 |
| Data type length   | 65000 |
| Character maximum length | 65000 |
| Numeric precision  |  |
| Numeric scale      |  |
| Datetime precision |  |
| Interval precision |  |
| Ordinal position   | 5 |

- [ RECORD 6 ]---------------------------------------------------
<p>| Table schema       | hcat |
| Hcatalog schema    | default |
| Table name         | hcatalogtypes |
| Is partition column| f |
| Column name        | boolcol |
| Hcatalog data type | boolean |</p>
<table>
<thead>
<tr>
<th>data_type</th>
<th>boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_type_id</td>
<td>5</td>
</tr>
<tr>
<td>data_type_length</td>
<td>1</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td>1</td>
</tr>
<tr>
<td>numeric_precision</td>
<td>2</td>
</tr>
<tr>
<td>numeric_scale</td>
<td>3</td>
</tr>
<tr>
<td>datetime_precision</td>
<td>4</td>
</tr>
<tr>
<td>interval_precision</td>
<td>5</td>
</tr>
<tr>
<td>ordinal_position</td>
<td>6</td>
</tr>
</tbody>
</table>

-[ RECORD 7 ]-_____________________________________
| table_schema       | hcat    |
| hcatalog_schema    | default |
| table_name         | hcatalogtypes |
| is_partition_column| f       |
| column_name        | timestampcol |
| hcatalog_data_type | string  |
| data_type          | varchar(65000) |
| data_type_id       | 9       |
| data_type_length   | 65000   |
| character_maximum_length | 65000 |
| numeric_precision  | 2       |
| numeric_scale      | 3       |
| datetime_precision | 4       |
| interval_precision | 5       |
| ordinal_position   | 7       |

-[ RECORD 8 ]-_____________________________________
| table_schema       | hcat    |
| hcatalog_schema    | default |
| table_name         | hcatalogtypes |
| is_partition_column| f       |
| column_name        | varbincol |
| hcatalog_data_type | binary  |
| data_type          | varbinary(65000) |
| data_type_id       | 17      |
| data_type_length   | 65000   |
| character_maximum_length | 65000 |
| numeric_precision  | 2       |
| numeric_scale      | 3       |
| datetime_precision | 4       |
| interval_precision | 5       |
| ordinal_position   | 8       |

-[ RECORD 9 ]-_____________________________________
| table_schema       | hcat    |
| hcatalog_schema    | default |
| table_name         | hcatalogtypes |
| is_partition_column| f       |
| column_name        | bincol   |
| hcatalog_data_type | binary  |
| data_type          | varbinary(65000) |
| data_type_id       | 17      |
| data_type_length   | 65000   |
| character_maximum_length | 65000 |
| numeric_precision  | 2       |
| numeric_scale      | 3       |
| datetime_precision | 4       |
| interval_precision | 5       |
| ordinal_position   | 9       |
See Also

- HCATALOG_SCHEMATA
- HCATALOG_TABLES
- HCATALOG_TABLE_LIST

HCATALOG_SCHEMATA

Lists all of the schemas defined using the HCatalog Connector. See Using the HCatalog Connector in Integrating with Apache Hadoop.

Unlike other HCatalog Connector-related system tables, this table makes no calls to Hive, so querying incurs very little overhead.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_ID</td>
<td>INTEGER</td>
<td>The Vertica Analytic Database ID number for the schema</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the schema defined in the Vertica Analytic Database catalog</td>
</tr>
<tr>
<td>SCHEMA_OWNER_ID</td>
<td>INTEGER</td>
<td>The ID number of the user who owns the Vertica Analytic Database schema</td>
</tr>
<tr>
<td>SCHEMA_OWNER</td>
<td>VARCHAR(128)</td>
<td>The username of the Vertica Analytic Database schema's owner</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMPTZ</td>
<td>The date and time the schema as created</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>VARCHAR(128)</td>
<td>The host name or IP address of the database server that holds the Hive metadata</td>
</tr>
<tr>
<td>PORT</td>
<td>INTEGER</td>
<td>The port number on which the metastore database listens for connections</td>
</tr>
<tr>
<td>HIVESERVER2_HOSTNAME</td>
<td>VARCHAR(128)</td>
<td>The host name or IP address of the HiveServer2 server for the Hive database</td>
</tr>
<tr>
<td>WEBSERVICE</td>
<td>VARCHAR(128)</td>
<td>The host name or IP address of the WebHCat server</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td></td>
<td>for the Hive database, if used</td>
</tr>
<tr>
<td>WEBSERVICE_PORT</td>
<td>INTEGER</td>
<td>The port number on which the WebHCat server listens for connections</td>
</tr>
<tr>
<td>WEBHDFS_ADDRESS</td>
<td>VARCHAR (128)</td>
<td>The host and port (&quot;host:port&quot;) for the WebHDFS service, used for reading ORC and Parquet files</td>
</tr>
<tr>
<td>HCATALOG_SCHEMA_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the schema or database in Hive to which the Vertica Analytic Database schema is mapped</td>
</tr>
<tr>
<td>HCATALOG_USER_NAME</td>
<td>VARCHAR(128)</td>
<td>The username the HCatalog Connector uses to authenticate itself to the Hive database.</td>
</tr>
<tr>
<td>HCATALOG_CONNECTION_TIMEOUT</td>
<td>INTEGER</td>
<td>The number of seconds the HCatalog Connector waits for a successful connection to the HiveServer or WebHCat server. A value of 0 means wait indefinitely.</td>
</tr>
<tr>
<td>HCATALOG_SLOW_TRANSFER_LIMIT</td>
<td>INTEGER</td>
<td>The lowest data transfer rate (in bytes per second) from the HiveServer2 or WebHCat server that the HCatalog Connector accepts.</td>
</tr>
<tr>
<td>HCATALOG_SLOW_TRANSFER_TIME</td>
<td>INTEGER</td>
<td>The number of seconds the HCatalog Connector waits before enforcing the data transfer rate lower limit by breaking the connection and terminating the query.</td>
</tr>
<tr>
<td>SSL_CONFIG</td>
<td>VARCHAR(128)</td>
<td>The path of the Hadoop ssl-client.xml configuration file, if using HiveServer2 with SSL wire encryption.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.
See Also

- **HCATALOG_COLUMNS**
- **HCATALOG_TABLE_LIST**
- **HCATALOG_TABLES**

**HCATALOG_TABLES**

Returns a detailed list of all tables made available through the HCatalog Connector. See [Using the HCatalog Connector](Integrating with Apache Hadoop) in Integrating with Apache Hadoop.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA_ID</td>
<td>INTEGER</td>
<td>ID number of the schema</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR(128)</td>
<td>The name of the Vertica Analytic Database schema through which the table is available</td>
</tr>
<tr>
<td>HCATALOG_SCHEMA</td>
<td>VARCHAR(128)</td>
<td>The name of the Hive schema or database that contains the table</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the table</td>
</tr>
<tr>
<td>HCATALOG_USER_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the HCatalog user whose credentials are used to access the table's data</td>
</tr>
<tr>
<td>MIN_FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The file size of the table's smallest data file, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>TOTAL_NUMBER_FILES</td>
<td>INTEGER</td>
<td>The number of files used to store this table's data in HDFS</td>
</tr>
<tr>
<td>LOCATION</td>
<td>VARCHAR(8192)</td>
<td>The URI for the directory containing this table's data, normally an HDFS URI</td>
</tr>
<tr>
<td>LAST_UPDATE_TIME</td>
<td>TIMESTAMPTZ</td>
<td>The last time data in this table was updated, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>VARCHAR(128)</td>
<td>The Hive SerDe class used to output data from this table</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
<td>table</td>
</tr>
<tr>
<td>LAST_ACCESS_TIME</td>
<td>TIMESTAMPTZ</td>
<td>The last time data in this table was accessed, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>MAX_FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The size of the largest data file for this table, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>IS_PARTITIONED</td>
<td>BOOLEAN</td>
<td>Whether this table is partitioned</td>
</tr>
<tr>
<td>PARTITION_EXPRESSION</td>
<td>VARCHAR(128)</td>
<td>The expression used to partition this table</td>
</tr>
<tr>
<td>TABLE_OWNER</td>
<td>VARCHAR(128)</td>
<td>The Hive user that owns this table in the Hive database, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>INPUT_FORMAT</td>
<td>VARCHAR(128)</td>
<td>The SerDe class used to read the data from this table</td>
</tr>
<tr>
<td>TOTAL_FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>Total number of bytes used by all of this table's data files</td>
</tr>
<tr>
<td>HCATALOG_GROUP</td>
<td>VARCHAR(128)</td>
<td>The permission group assigned to this table, if using WebHCat; null if using HiveServer2</td>
</tr>
<tr>
<td>PERMISSION</td>
<td>VARCHAR(128)</td>
<td>The Unix file permissions for this group, as shown by the ls -l command, if using WebHCat; null if using HiveServer2</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

**See Also**

- `HCATALOG_SCHEMA
- `HCATALOG_COLUMNS

Vertica Analytic Database (9.0.x)
HCATALOG_TABLE_LIST

A concise list of all tables contained in all Hive schemas and databases available through the HCatalog Connector. See Using the HCatalog Connector in Integrating with Apache Hadoop.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA_ID</td>
<td>INTEGER</td>
<td>Internal ID number for the schema containing the table</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR(128)</td>
<td>Name of the Vertica Analytic Database schema through which the table is available</td>
</tr>
<tr>
<td>HCATALOG_SCHEMA</td>
<td>VARCHAR(128)</td>
<td>Name of the Hive schema or database containing the table</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the table</td>
</tr>
<tr>
<td>HCATALOG_USER_NAME</td>
<td>VARCHAR(128)</td>
<td>Name of Hive user used to access the table</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit permissions are required; however, users see only the records that correspond to schemas they have permissions to access.

**Notes**

- Querying this table results in one call to HiveServer2 for each Hive schema defined using the HCatalog Connector. This means that the query usually takes longer than querying other system tables.

- Querying this table is faster than querying HCATALOG_TABLES. Querying HCATALOG_TABLE_LIST only makes one HiveServer2 call per HCatalog schema versus one call per table for HCATALOG_TABLES.
Example

The following example demonstrates defining a new HCatalog schema then querying HCATALOG_TABLE_LIST. Note that one table defined in a different HCatalog schema also appears. HCATALOG_TABLE_LIST lists all of the tables available in any of the HCatalog schemas:

```
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost'
- HCATALOG_SCHEMA='default' HCATALOG_DB='default' HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> \
Expanded display is on.
=> SELECT * FROM v_catalog.hcatalog_table_list;
- [ RECORD 1 ]------------------------
table_schema_id       | 45035996273748980
table_schema          | hcat
hcatalog_schema       | default
table_name            | weblogs
hcatalog_user_name    | hcatuser
- [ RECORD 2 ]------------------------
table_schema_id       | 45035996273748980
table_schema          | hcat
hcatalog_schema       | default
table_name            | tweets
hcatalog_user_name    | hcatuser
- [ RECORD 3 ]------------------------
table_schema_id       | 45035996273748980
table_schema          | hcat
hcatalog_schema       | default
table_name            | messages
hcatalog_user_name    | hcatuser
- [ RECORD 4 ]------------------------
table_schema_id       | 45035996273864948
table_schema          | hiveschema
hcatalog_schema       | default
table_name            | weblogs
hcatalog_user_name    | hcatuser
```

See Also

- HCATALOG_COLUMNS
- HCATALOG_SCHEMATA
- HCATALOG_TABLES

KEYWORDS

Identifies Vertica reserved and non-reserved keywords.
### Keywords

Vertica reserved or non-reserved keyword.

**Reserved**

Indicates whether a keyword is reserved or non-reserved:

- **R**: reserved
- **N**: non-reserved

### LARGE_CLUSTER_CONFIGURATION_STATUS

Shows the current cluster nodes and control node (spread hosts) designations in the Catalog so you can see if they match.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node in the cluster.</td>
</tr>
<tr>
<td>SPREAD_HOST_NAME</td>
<td>VARCHAR</td>
<td>The host name of the control node (the host that manages control message responsibilities)</td>
</tr>
<tr>
<td>CONTROL_NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the control node</td>
</tr>
</tbody>
</table>

### See Also

**Large Cluster** in the Administrator's Guide

### LICENSE_AUDITS

Lists the results of Vertica's license automatic compliance audits. See How Vertica Calculates Database Size in the Administrator's Guide.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The estimated raw data size of the database</td>
</tr>
<tr>
<td>LICENSE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The licensed data allowance</td>
</tr>
<tr>
<td>USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Percentage of the licensed allowance used</td>
</tr>
<tr>
<td>AUDIT_START_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>When the audit started</td>
</tr>
<tr>
<td>AUDIT_END_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>When the audit finished</td>
</tr>
<tr>
<td>CONFIDENCE_LEVEL_PERCENT</td>
<td>FLOAT</td>
<td>The confidence level of the size estimate</td>
</tr>
<tr>
<td>ERROR_TOLERANCE_PERCENT</td>
<td>FLOAT</td>
<td>The error tolerance used for the size estimate</td>
</tr>
<tr>
<td>USED_SAMPLING</td>
<td>BOOLEAN</td>
<td>Whether data was randomly sampled (if false, all of the data was analyzed)</td>
</tr>
<tr>
<td>CONFIDENCE_INTERVAL_LOWER_BOUNDgetBytes</td>
<td>INTEGER</td>
<td>The lower bound of the data size estimate within the confidence level</td>
</tr>
<tr>
<td>CONFIDENCE_INTERVAL_UPPER_BOUNDgetBytes</td>
<td>INTEGER</td>
<td>The upper bound of the data size estimate within the confidence level</td>
</tr>
<tr>
<td>SAMPLE_COUNT</td>
<td>INTEGER</td>
<td>The number of data samples used to generate the estimate</td>
</tr>
<tr>
<td>CELL_COUNT</td>
<td>INTEGER</td>
<td>The number of cells in the database</td>
</tr>
<tr>
<td>AUDITED_DATA</td>
<td>VARCHAR</td>
<td>The type of data audited, which includes regular (non-flex), flex, and total data</td>
</tr>
</tbody>
</table>

**LICENCES**

For all licenses, provides information on license types, the dates for which licenses are valid, and the limits the licenses impose.
### Column Name | Data Type | Description
---|---|---
LICENSE_ID | INTEGER | Unique numeric ID assigned by the Vertica catalog, which identifies the license.
NAME | VARCHAR | The license’s name. (The license name in this column could be represented by a long license key.)
LICENSEE | VARCHAR | The entity to which the product is licensed.
START_DATE | VARCHAR | The start date for which the license is valid.
END_DATE | VARCHAR | The end date until which the license is valid (or "Perpetual" if the license has no expiration).
LICENSETYPE | VARCHAR | The type of the license (for example, Premium Edition).
PARENT | VARCHAR | The parent license (field is blank if there is no parent).
SIZE | VARCHAR | The size limit for data on the license.
IS_SIZE_LIMIT_ENFORCED | BOOLEAN | Indicates whether the license includes enforcement of data and node limits, where t is true and f is false.
NODE_RESTRICTION | VARCHAR | The node limit the license imposes.
CONFIGURED_ID | INTEGER | A long license key.

### MATERIALIZE_FLEXTABLE_COLUMNS_RESULTS

Returns the results after you run the flex table function, **MATERIALIZE_FLEXTABLE_COLUMNS**. The system table includes the following information:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the license.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The license’s name.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The Vertica product toward which the license is applied.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>CREATION_TIME</td>
<td>VARCHAR</td>
<td>The start date for which the license is valid.</td>
</tr>
<tr>
<td>KEY_NAME</td>
<td>VARCHAR</td>
<td>The end date until which the license is valid (or &quot;Perpetual&quot; if the license has no expiration).</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR</td>
<td>The function status, which can be one of these values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ADDED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EXISTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>BOOLEAN</td>
<td>The message associated with the status in the previous column, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Column of same name already exists in table definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add operation failed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No data type guess provided to add column</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT table_name, creation_time, key_name, status, message
FROM v_catalog.materialize_flextable_columns_results WHERE table_name = 'twitter_r';
```

```
table_name | creation_time | key_name | status | message
----------|--------------|----------|--------|-------------------------
twitter_r | 2013-11-20 17:00:27.945484-05 | contributors | ADDED | Added successfully
twitter_r | 2013-11-20 17:00:27.945511-05 | entities.hashtags | ADDED | Added successfully
twitter_r | 2013-11-20 17:00:27.945519-05 | entities.urls | ADDED | Added successfully
twitter_r | 2013-11-20 17:00:27.945532-05 | created_at | EXISTS | Column of same name already exists in table definition
(4 rows)
```
MODELS

Lists details about the machine-learning models in the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_ID</td>
<td>INTEGER</td>
<td>The model's internal ID.</td>
</tr>
<tr>
<td>MODEL_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the model.</td>
</tr>
<tr>
<td>SCHEMA_ID</td>
<td>INTEGER</td>
<td>The schema's internal ID.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the schema.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>The model owner's ID.</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR(128)</td>
<td>The user who created the model.</td>
</tr>
<tr>
<td>CATEGORY</td>
<td>VARCHAR(128)</td>
<td>The type of model. By default, models created in Vertica are assigned to the Vertica_Models category.</td>
</tr>
<tr>
<td>MODEL_TYPE</td>
<td>VARCHAR(128)</td>
<td>The type of algorithm used to create the model.</td>
</tr>
<tr>
<td>IS_COMPLETE</td>
<td>VARCHAR(128)</td>
<td>Denotes whether the model is complete and ready for use in machine learning functions. This field is usually false when the model is being trained. Once the training is complete, the field is set to true.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMPTZ</td>
<td>The time the model was created.</td>
</tr>
<tr>
<td>SIZE</td>
<td>INTEGER</td>
<td>The size of the model in bytes.</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT * FROM models;
- [ RECORD 1 ]-------------------------
model_id | 45035996273714020
model_name | myLinearRegModel
schema_id | 45035996273704980
schema_name | public
owner_id | 45035996273704962
owner_name | dbadmin
category | VERTICA_MODELS
model_type | LINEAR_REGRESSION
is_complete | t
```
NODES

Lists details about the nodes in the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR(128)</td>
<td>The name of the node.</td>
</tr>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the node.</td>
</tr>
<tr>
<td>NODE_STATE</td>
<td>VARCHAR(128)</td>
<td>The node's current state (up, down, recovering, etc.).</td>
</tr>
<tr>
<td>NODE_ADDRESS</td>
<td>VARCHAR(80)</td>
<td>The host address of the node.</td>
</tr>
<tr>
<td>NODE_ADDRESS_FAMILY</td>
<td>VARCHAR(10)</td>
<td>The IP Version of the node_address. For example, ipv4.</td>
</tr>
<tr>
<td>EXPORT_ADDRESS</td>
<td>VARCHAR(8192)</td>
<td>The IP address of the node (on the public network) used for import/export operations and native load-balancing.</td>
</tr>
<tr>
<td>EXPORT_ADDRESS_FAMILY</td>
<td>VARCHAR(10)</td>
<td>The IP Version of the export_address. For example, ipv4.</td>
</tr>
<tr>
<td>CATALOG_PATH</td>
<td>VARCHAR(8192)</td>
<td>The absolute path to the catalog on the node.</td>
</tr>
<tr>
<td>NODE_TYPE</td>
<td>VARCHAR(9)</td>
<td>The type of the node. For more information on the types of nodes, refer to Setting Node Type.</td>
</tr>
<tr>
<td>IS_EPHEMERAL</td>
<td>BOOLEAN</td>
<td>(Deprecated) True if this node has been marked as ephemeral. (in preparation for removing it from the cluster).</td>
</tr>
<tr>
<td>STANDING_IN_FOR</td>
<td>VARCHAR(128)</td>
<td>The name of the node that this node is currently replacing.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LAST_MSG_FROM_NODE_AT</td>
<td>TIMESTAMPZ</td>
<td>The date and time the last message was received from this node.</td>
</tr>
<tr>
<td>NODE_DOWN_SINCE</td>
<td>TIMESTAMPZ</td>
<td>The amount of time that the replaced node has been unavailable.</td>
</tr>
</tbody>
</table>

**ODBC_COLUMNS**

Provides table column information. The format is defined by the ODBC standard for the ODBC SQLColumns metadata. Details on the ODBC SQLColumns format are available in the ODBC specification: http://msdn.microsoft.com/en-us/library/windows/desktop/ms711683%28v=vs.85%29.aspx.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema in which the column resides. If the column does not reside in a schema, this field is empty.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the table in which the column resides.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>INTEGER</td>
<td>The data type of the column. This can be an ODBC SQL data type or a driver-specific SQL data type. This column corresponds to the ODBC_TYPE column in the TYPES table.</td>
</tr>
<tr>
<td>DATA_TYPE_NAME</td>
<td>VARCHAR</td>
<td>The driver-specific data type name.</td>
</tr>
<tr>
<td>COLUMN_SIZE</td>
<td>INTEGER</td>
<td>The ODBC-defined data size of the column.</td>
</tr>
<tr>
<td>BUFFER_LENGTH</td>
<td>INTEGER</td>
<td>The transfer octet length of a column is the maximum number of bytes returned to the application when data is transferred to its default C data type. See <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/ms713979%28v=vs.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/ms713979%28v=vs.85%29.aspx</a></td>
</tr>
<tr>
<td>DECIMAL_DIGITS</td>
<td>INTEGER</td>
<td>The total number of significant digits to the right of the decimal point. This value has no meaning for non-decimal data types.</td>
</tr>
</tbody>
</table>
## NUM_PREC_RADIX

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>The radix Vertica reports decimal_digits and columns_size as. This value is always 10, because it refers to a number of decimal digits, rather than a number of bits.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Indicates whether the column can contain null values. Values are 0 or 1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>The textual remarks for the column.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>The default value of the column.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>The SQL data type of the column.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>The subtype for a datetime data type. This value has no meaning for non-datetime data types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>The maximum length of a string or binary data column.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Indicates the position of the column in the table definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>Values can be YES or NO, determined by the value of the NULLABLE column.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Indicates whether the column is a sequence, for example, an auto-increment column.</td>
</tr>
</tbody>
</table>

## PASSWORD_AUDITOR

Stores information about individual users and their password information. This table also indicates if users are using hash authentication, which is the associated security algorithm.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Unique ID for the user.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user.</td>
</tr>
<tr>
<td>ACCTEXPIRED</td>
<td>BOOLEAN</td>
<td>Indicates if the user’s password expires. 'f' indicates that it does not expire. 't' indicates that it does expire.</td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x)
### Column Name | Data Type | Description
---|---|---
SECURITY_ALGORITHM | VARCHAR | User-level security algorithm for hash authentication. **Valid values:**
- 'NONE' (Default. System-level security algorithm is used.)
- 'MD5'
- 'SHA512'

SYSTEM_SECURITY_ALGORITHM | VARCHAR | System-level security algorithm for hash authentication. **Default value:**
- 'NONE' (Uses MD5 algorithm.) **Valid values:**
- 'MD5'
- 'SHA512'

EFFECTIVE_SECURITY_ALGORITHM | VARCHAR | The resulting security algorithm, depending on the values of SECURITY_ALGORITHM and SYSTEM_SECURITY_ALGORITHM.

---

## PASSWORDS
Contains user passwords information. This table stores current passwords and past passwords if any Profiles have PASSWORD_REUSE_TIME or PASSWORD_REUSE_MAX parameters set. See CREATE PROFILE for details.

### Column Name | Data Type | Description
---|---|---
USER_ID | INTEGER | The ID of the user who owns the password.
USER_NAME | VARCHAR | The name of the user who owns the password.
PASSWORD | VARCHAR | The encrypted password.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD_CREATE_TIME</td>
<td>DATETIME</td>
<td>The date and time when the password was created.</td>
</tr>
<tr>
<td>IS_CURRENT_PASSWORD</td>
<td>BOOLEAN</td>
<td>Denotes whether this is the user's current password. Non-current passwords are retained to enforce password reuse limitations.</td>
</tr>
<tr>
<td>PROFILE_ID</td>
<td>INTEGER</td>
<td>The ID number of the profile to which the user is assigned.</td>
</tr>
<tr>
<td>PROFILE_NAME</td>
<td>VARCHAR</td>
<td>The name of the profile to which the user is assigned.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_MAX</td>
<td>VARCHAR</td>
<td>The number password changes that must take place before an old password can be reused.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_TIME</td>
<td>VARCHAR</td>
<td>The amount of time that must pass before an old password can be reused.</td>
</tr>
</tbody>
</table>

**PRIMARY_KEYS**

Provides primary key information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the constraint.</td>
</tr>
<tr>
<td>CONSTRAINT_NAME</td>
<td>VARCHAR</td>
<td>The constraint name for which information is listed.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The column name for which information is listed.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>VARCHAR</td>
<td>The position of the column within the key. The numbering of columns starts at 1.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name for which information is listed.</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>VARCHAR</td>
<td>The constraint type, p, for primary key.</td>
</tr>
<tr>
<td>IS_ENABLED</td>
<td>BOOLEAN</td>
<td>Indicates if a table column constraint for a PRIMARY KEY is enabled by default. Can be t (True) or f (False).</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
</tbody>
</table>
# PROFILE_PARAMETERS

Defines what information is stored in profiles.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>INTEGER</td>
<td>The ID of the profile to which this parameter belongs.</td>
</tr>
<tr>
<td>PROFILE_NAME</td>
<td>VARCHAR</td>
<td>The name of the profile to which this parameter belongs.</td>
</tr>
<tr>
<td>PARAMETER_TYPE</td>
<td>VARCHAR</td>
<td>The policy type of this parameter (password_complexity, password_security, etc.)</td>
</tr>
<tr>
<td>PARAMETER_NAME</td>
<td>VARCHAR</td>
<td>The name of the parameter.</td>
</tr>
<tr>
<td>PARAMETER_LIMIT</td>
<td>VARCHAR</td>
<td>The parameter's value.</td>
</tr>
</tbody>
</table>

# PROFILES

Provides information about password policies that you set using the CREATE PROFILE statement.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROFILE_ID</td>
<td>INTEGER</td>
<td>Unique identifier for the profile.</td>
</tr>
<tr>
<td>PROFILE_NAME</td>
<td>VARCHAR</td>
<td>Profile name.</td>
</tr>
<tr>
<td>PASSWORD_LIFE_TIME</td>
<td>VARCHAR</td>
<td>Number of days before the user's password expires. After expiration, the user is forced to change passwords during login or warned that their password has expired if password_grace_time is set to a value other than zero or unlimited.</td>
</tr>
<tr>
<td>PASSWORD_GRACE_TIME</td>
<td>VARCHAR</td>
<td>Number of days users are allowed to log in after their passwords expire. During the grace time, users are warned about their expired passwords when they log in. After the grace period, the user is forced to change passwords if he or she hasn't already.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PASSWORD_REUSE_MAX</td>
<td>VARCHAR</td>
<td>Number of password changes that must occur before the current password can be reused.</td>
</tr>
<tr>
<td>PASSWORD_REUSE_TIME</td>
<td>VARCHAR</td>
<td>Number of days that must pass after setting a password before it can be used again.</td>
</tr>
<tr>
<td>FAILED_LOGIN_ATTEMPTS</td>
<td>VARCHAR</td>
<td>Number of consecutive failed login attempts that triggers Vertica to lock the account.</td>
</tr>
<tr>
<td>PASSWORD_LOCK_TIME</td>
<td>VARCHAR</td>
<td>Number of days an account is locked after being locked due to too many failed login attempts.</td>
</tr>
<tr>
<td>PASSWORD_MAX_LENGTH</td>
<td>VARCHAR</td>
<td>Maximum number of characters allowed in a password.</td>
</tr>
<tr>
<td>PASSWORD_MIN_LENGTH</td>
<td>VARCHAR</td>
<td>Minimum number of characters required in a password.</td>
</tr>
<tr>
<td>PASSWORD_MIN_LETTERS</td>
<td>VARCHAR</td>
<td>The minimum number of letters (either uppercase or lowercase) required in a password.</td>
</tr>
<tr>
<td>PASSWORD_MIN_LOWERCASE_LETTERS</td>
<td>VARCHAR</td>
<td>The minimum number of lowercase.</td>
</tr>
<tr>
<td>PASSWORD_MIN_UPPERCASE_LETTERS</td>
<td>VARCHAR</td>
<td>The minimum number of uppercase letters required in a password.</td>
</tr>
<tr>
<td>PASSWORD_MIN_DIGITS</td>
<td>VARCHAR</td>
<td>The minimum number of digits required in a password.</td>
</tr>
<tr>
<td>PASSWORD_MIN_SYMBOLS</td>
<td>VARCHAR</td>
<td>The minimum of symbols (for example, !, #, $, etc.) required in a password.</td>
</tr>
</tbody>
</table>

**Notes**

Non-superusers querying this table see only the information for the profile to which they are assigned.
### See Also
- [CREATE PROFILE](#)
- [ALTER PROFILE](#)

### PROJECTION_CHECKPOINT_EPOCHS

Provides details on when checkpoint epochs are.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>Unique numeric identifier of this projection's node.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of this projection's node.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA_ID</td>
<td>INTEGER</td>
<td>Unique numeric identifier of the projection schema.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the projection schema.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric identifier of this projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Name of this projection.</td>
</tr>
<tr>
<td>IS_UP_TO_DATE</td>
<td>BOOLEAN</td>
<td>Specifies whether the projection is <strong>up to date</strong> and available to participate in query execution.</td>
</tr>
<tr>
<td>CHECKPOINT_EPOCH</td>
<td>INTEGER</td>
<td>Checkpoint epoch of the projection on the corresponding node. Data up to and including this epoch is in persistent storage, and can be recovered in the event of node failure.</td>
</tr>
<tr>
<td>WOULD_RECOVER</td>
<td>BOOLEAN</td>
<td>Determines whether data up to and including CHECKPOINT_EPOCH can be used to <strong>recover from an unclean shutdown</strong>:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- t: CHECKPOINT_EPOCH is less than or equal to the cluster's Last Good Epoch, so data up to and including this epoch can be used during recovery.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IS_BEHIND_AHM</td>
<td>BOOLEAN</td>
<td>Specifies whether CHECKPOINT_EPOCH is less than the AHM (ancient history mark). If set to t (true), data for this projection cannot rolled back.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See also: GET_AHM_EPOCH</td>
</tr>
</tbody>
</table>

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Examples

```sql
=> SELECT epoch FROM t;
epoch
-------
 52
 52
 53
(3 rows)

=> SELECT node_name, projection_schema, projection_name, is_up_to_date, checkpoint_epoch
FROM projection_checkpoint_epochs;
node_name          | projection_schema | projection_name | is_up_to_date | checkpoint_epoch
--------------------|-------------------|-----------------|---------------|-----------------|
 v_vmart_node0001  | public            | t_super         | t             | 51              |
 v_vmart_node0001  | public            | p_super         | t             | 51              |
(2 rows)

=> SELECT DO_TM_TASK('moveout', '');
  do_tm_task

Task: moveout
(Table: public.t) (Projection: public.t_super)
(Table: public.p) (Projection: public.p_super)
(1 row)

=> SELECT node_name, projection_schema, projection_name, is_up_to_date, checkpoint_epoch
FROM projection_checkpoint_epochs;
node_name          | projection_schema | projection_name | is_up_to_date | checkpoint_epoch
--------------------|-------------------|-----------------|---------------|-----------------|
 v_vmart_node0001  | public            | t_super         | t             | 53              |
```

See: GET_LAST_GOOD_EPOCH
## PROJECTION_COLUMNS

Provides information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The projection column name.</td>
</tr>
<tr>
<td>COLUMN_POSITION</td>
<td>INTEGER</td>
<td>The ordinal position of a projection's column used in the <code>CREATE_PROJECTION</code> statement.</td>
</tr>
<tr>
<td>SORT_POSITION</td>
<td>INTEGER</td>
<td>The projection's column sort specification, as specified in <code>CREATE_PROJECTION</code> .. ORDER BY clause. If the column is not included in the projection's sort order, <code>SORT_POSITION</code> output is NULL.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>INTEGER</td>
<td>A unique numeric object ID (OID) that identifies the associated projection column object and is assigned by the Vertica catalog. This field is helpful as a key to other system tables.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR</td>
<td>Matches the corresponding table column data type (see <code>V_CATALOG.COLUMNS</code>). <code>DATA_TYPE</code> is provided as a complement to <code>ENCODING_TYPE</code>.</td>
</tr>
</tbody>
</table>
| ENCODING_TYPE          | VARCHAR   | The encoding type defined on the
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>projection column.</td>
</tr>
<tr>
<td>ACCESS_RANK</td>
<td>INTEGER</td>
<td>The access rank of the projection column. See the ACCESSRANK parameter in the CREATE PROJECTION statement for more information.</td>
</tr>
<tr>
<td>GROUP_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID (OID) that identifies the group and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema in which the projection is stored.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name that contains the projection.</td>
</tr>
<tr>
<td>TABLE_COLUMN_ID</td>
<td>VARCHAR</td>
<td>A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.</td>
</tr>
<tr>
<td>TABLE_COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The projection's corresponding table column name.</td>
</tr>
<tr>
<td>STATISTICS_TYPE</td>
<td>VARCHAR</td>
<td>The type of statistics the column contains:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- NONE: No statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ROWCOUNT: Created from existing catalog metadata, which Vertica automatically and periodically updates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FULL: Created by running ANALYZE_STATISTICS</td>
</tr>
<tr>
<td>STATISTICS_UPDATED_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>The time at which the columns statistics were last updated. By querying this column, along with STATISTICS_TYPE and PROJECTION_COLUMN_NAME, you can</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>identify projection columns whose statistics need updating. See also V_CATALOG.PROJECTIONS.HAS_STATISTICS.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS_EXPRESSION</td>
<td>BOOLEAN</td>
<td>Indicates whether this projection column is calculated with an expression. For aggregate columns, IS_EXPRESSION is always true.</td>
</tr>
<tr>
<td>IS_AGgregate</td>
<td>BOOLEAN</td>
<td>Indicates whether the column is an aggregated column in a live aggregate projection. IS_AGREGATE is always false for Top-K projection columns.</td>
</tr>
<tr>
<td>PARTITION_BY_POSITION</td>
<td>INTEGER</td>
<td>Position of that column in the PARTITION BY and GROUP BY clauses, if applicable.</td>
</tr>
<tr>
<td>ORDER_BY_POSITION</td>
<td>INTEGER</td>
<td>Set only for Top-K projections, specifies the column's position in the ORDER BY clause, as defined in the projection definition's window partition clause. If the column is omitted from the ORDER BY clause, ORDER_BY_POSITION output is NULL.</td>
</tr>
<tr>
<td>ORDER_BY_TYPE</td>
<td>INTEGER</td>
<td>Type of sort order:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ASC NULLS FIRST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ASC NULLS LAST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DESC NULLS FIRST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DESC NULLS LAST</td>
</tr>
<tr>
<td>COLUMN_EXPRESSION</td>
<td>VARCHAR</td>
<td>Expression that calculates the column value.</td>
</tr>
</tbody>
</table>

**Examples**

See Determining When Statistics Were Last Updated.
See Also

- PROJECTIONS
- ANALYZE_STATISTICS
- CREATE PROJECTION
- Collecting Database Statistics

PROJECTION_DELETE_CONCERNS

Lists projections whose design may cause performance issues when deleting data. This table is generated by calling the EVALUATE_DELETE_PERFORMANCE function. See DELETE and UPDATE Optimization in the Administrator's Guide for more information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>The ID number of the projection</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema containing the projection</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection's name</td>
</tr>
<tr>
<td>CREATION_TIME</td>
<td>TIMESTAMPTZ</td>
<td>When the projection was created</td>
</tr>
<tr>
<td>LAST_MODIFIED_TIME</td>
<td>TIMESTAMPTZ</td>
<td>When the projection was last modified</td>
</tr>
<tr>
<td>COMMENT</td>
<td>VARCHAR</td>
<td>A comment describing the potential delete performance issue.</td>
</tr>
</tbody>
</table>

PROJECTIONS

Provides information about projections.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECTION_SCHEMA_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the specific schema that contains the projection and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema that contains the projection.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the projection and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_BASENAME</td>
<td>VARCHAR</td>
<td>The base name used for other projections. For auto-created projections, <code>projection_basename</code> is identical to the <code>anchor_table_name</code>. For a manually-created projection, <code>projection_basename</code> is the name you supply.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the projection owner and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR</td>
<td>The name of the projection's owner.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_ID</td>
<td>INTEGER</td>
<td>For pre-join projections, the unique numeric identification (OID) of the anchor table. If the projection is not a pre-join projection, this value is the OID of the table from which the projection was created. A projection has only one anchor (fact) table.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>For pre-join projections, the name of the anchor table. If the projection is not a pre-join projection, the name of the table from which the projection was created.</td>
</tr>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID (OID) for any nodes that contain any unsegmented projections.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The names of any nodes that contain the projection. This column returns information for unsegmented projections only.</td>
</tr>
<tr>
<td>IS_PREJOIN</td>
<td>BOOLEAN</td>
<td>Specifies whether the projection is a pre-join projection.</td>
</tr>
<tr>
<td>CREATED_EPOCH</td>
<td>INTEGER</td>
<td>The epoch in which the projection was created.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CREATE_TYPE</td>
<td>VARCHAR</td>
<td>The method in which the projection was created:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CREATE_PROJECTION: A custom projection created using a CREATE_PROJECTION statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CREATE_TABLE: A superprojection that was automatically created when its associated table was created using a CREATE_TABLE statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ALTER_TABLE: The system automatically created the key projection in response to a non-empty table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CREATE_TABLE WITH PROJ_CLAUSE: A superprojection created using a CREATE_TABLE statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DELAYED_CREATION: A superprojection that was automatically created when data was loaded into its associated table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DESIGNER: A new projection created by the Database Designer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SYSTEM_TABLE: A projection that was automatically created for a system table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rebalancing does not change the CREATE_TYPE value for a projection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERIFIED_FAULT_TOLERANCE</td>
<td>INTEGER</td>
<td>The projection K-safe value. This value can be greater than the database K-safety value (if more replications of a projection exist than are required to meet the database K-safety). This value cannot be less than the database K-safe setting.</td>
</tr>
<tr>
<td>IS_UP_TO_DATE</td>
<td>BOOLEAN</td>
<td>Specifies whether projection data is up to date. Only up-to-date projections are available to participate in query execution.</td>
</tr>
<tr>
<td>HAS_STATISTICS</td>
<td>BOOLEAN</td>
<td>Specifies whether there are statistics for any column in the projection:</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• This column returns true only when all non-epoch columns for a table have full statistics. Otherwise, the column returns false. See <strong>ANALYZE_STATISTICS</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Projections that have no data never have full statistics. Use system table <strong>PROJECTION_STORAGE</strong> to determine whether your projection contains data.</td>
</tr>
<tr>
<td>IS_SEGMENTED</td>
<td>BOOLEAN</td>
<td>Specifies whether the projection is segmented.</td>
</tr>
<tr>
<td>SEGMENT_EXPRESSION</td>
<td>VARCHAR</td>
<td>The segmentation expression used for the projection. In the following example for the clicks_agg projection, the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hash(clicks.user_id, (clicks.click_time)::date)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>indicate that the projection was created with the following expression:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEGMENTED BY HASH( clicks.user_id, (clicks.click_time)::date)</td>
</tr>
<tr>
<td>SEGMENT_RANGE</td>
<td>VARCHAR</td>
<td>The percentage of projection data stored on each node, according to the segmentation expression. For example, segmenting a projection by the <strong>HASH</strong> function on all nodes results in a SEGMENT_RANGE value such as the following:</td>
</tr>
</tbody>
</table>
|                             |                 | implicit range: v_testcr_node0005[33.3%]  
v_testcr_node0006[33.3%]  
v_testcr_node0004[33.3%]                                                                                                       |
<p>| IS_SUPER_PROJECTION         | BOOLEAN         | Specifies whether a projection is a superprojection.                                                                                           |
| IS_KEYCONSTRAINT_PROJECTION | BOOLEAN         | Indicates whether a projection is a key constraint projection:                                                                               |
|                             |                 | • t: A key constraint projection that validates a key constraint. (Vertica uses the projection to efficiently enforce at least one enabled key constraint.) |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAS_EXPRESSIONS</td>
<td>BOOLEAN</td>
<td>Specifies whether this projection has expressions that define the column values. HAS_EXPRESSIONS is always true for live aggregate projections.</td>
</tr>
<tr>
<td>IS_AMBERATE_PROJECTION</td>
<td>BOOLEAN</td>
<td>Specifies whether this projection is a live aggregate projection.</td>
</tr>
<tr>
<td>AGGREGATE_TYPE</td>
<td>VARCHAR</td>
<td>Specifies the type of live aggregate projection:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GROUPBY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TOPK</td>
</tr>
<tr>
<td>IS_SHARED</td>
<td>BOOLEAN</td>
<td>Indicates whether the projection is located on shared storage.</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT * FROM projections;
```

```sql
- [ RECORD 1 ]-+------------------------------------------------------------------------------------------------------------------
| projection_schema_id | 45035996273704980      |
| projection_schema    | public                 |
| projection_id        | 45035996273707840      |
| projection_name      | date_dimension_b0      |
| projection_base_name | date_dimension         |
| owner_id             | 45035996273704962      |
| owner_name           | dbadmin                |
| anchor_table_id      | 45035996273707756      |
| anchor_table_name    | date_dimension         |
| node_id              | 0                      |
| node_name            |                         |
| is_prejoin           | f                      |
| created_epoch        | 17                     |
| create_type          | DELAYED CREATION       |
| verified_fault_tolerence | 1                 |
| is_up_to_date        | t                      |
| has_statistics       | f                      |
| is_segmented         | t                      |
| segment_expression   | hash(date_dimension.date_key) |
| segment_range        | implicit range: v_vmart_node0001[33.3%] v_vmart_node0002[33.3%] v_vmart_node0003[33.3%] |
| is_super_projection  | t                      |
| is_key_constraint_projection | f                 |
| has_expressions      | f                      |
| is_aggregate_projection | f                    |
| aggregate_type       |                         |
See Also

PROJECTION_COLUMNS

RESOURCE_POOL_DEFAULTS

Provides information about the default parameter values for built-in and user-defined resource pools. You can use ALTER RESOURCE_POOL to restore default parameter values for any resource pool by setting that parameter to DEFAULT. To see the default values for the built-in resource pools, refer to Built-In Pool Configuration.

Privileges

None

See Also

- RESOURCE_POOLS

RESOURCE_POOLS

Displays information about the parameters specified for the resource pool by CREATE RESOURCE_POOL or ALTER RESOURCE_POOL.

Note: Column names in the RESOURCE_POOL table mirror syntax in the CREATE RESOURCE POOL statement. Therefore, column names do not use underscores.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>VARCHAR</td>
<td>The name of the resource pool.</td>
</tr>
<tr>
<td>IS_INTERNAL</td>
<td>BOOLEAN</td>
<td>Denotes whether a pool is one of the Built-In Pools.</td>
</tr>
<tr>
<td>MEMORYSIZE</td>
<td>VARCHAR</td>
<td>Value of the amount of memory</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allocated to the resource pool.</td>
</tr>
<tr>
<td>MAXMEMORYSIZE</td>
<td>VARCHAR</td>
<td>Value assigned as the maximum size the resource pool could grow by borrowing memory from the GENERAL pool.</td>
</tr>
<tr>
<td>EXECUTIONPARALLELISM</td>
<td>INTEGER</td>
<td>Limits the number of threads used to process any single query issued in this resource pool.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>INTEGER</td>
<td>Value of PRIORITY parameter specified when defining the pool.</td>
</tr>
</tbody>
</table>
| RUNTIMEPRIORITY         | VARCHAR         | Value that indicates the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are:  
  - HIGH  
  - MEDIUM (default)  
  - LOW  
These values are relative to each other. Queries with a HIGH run-time priority are given more CPU and I/O resources than those with a MEDIUM or LOW run-time priority. |
<p>| RUNTIMEPRIORITYTHRESHOLD| INTEGER         | Limits in seconds how soon a query must finish before the Resource Manager assigns to it the resource pool's RUNTIMEPRIORITY setting.                                                                 |
| QUEUETIMEOUT            | INTEGER or INTERVAL | The interval or value in seconds of the QUEUETIMEOUT parameter specified when defining the pool. Represents the maximum amount of time the request is allowed to wait for resources to become available before being rejected. |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANNEDCONCURRENCY</td>
<td>INTEGER</td>
<td>Value of PLANNEDCONCURRENCY parameter specified when defining the pool, which represents the preferred number of concurrently executing queries in the resource pool.</td>
</tr>
<tr>
<td>MAXCONCURRENCY</td>
<td>INTEGER</td>
<td>Value of MAXCONCURRENCY parameter specified when defining the pool, which represents the maximum number of concurrent execution slots available to the resource pool.</td>
</tr>
<tr>
<td>RUNTIMECAP</td>
<td>INTERVAL</td>
<td>The maximum time a query in the pool can execute.</td>
</tr>
<tr>
<td>SINGLEINITIATOR</td>
<td>BOOLEAN</td>
<td>Specifies whether all requests using this pool are issued against the same initiator node or multiple initiator nodes can be used. Included for backwards compatibility. For all built-in resource pools, this must be set to false for all user-defined pools.</td>
</tr>
<tr>
<td>CPUAFFINITYSET</td>
<td>VARCHAR</td>
<td>Value which represents the set of CPUs on which queries associated with this pool are executed. For example, '0, 2-4' for the 0, 2, 3, and 4 CPUs, or '25%' for a percentage of available CPUs. Percentage values are rounded down to whole CPUs.</td>
</tr>
<tr>
<td>CPUAFFINITYMODE</td>
<td>VARCHAR</td>
<td>The mode of the CPU affinity. If CPUAFFINITYSET is set to a CPU percentage or index/index list, then its value is one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SHARED: Pool is pinned to the CPU indexes or percentage defined in CPUAFFINITYSET, and other pools can share the same CPUs</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
**EXCLUSIVE** | | EXCLUSIVE: Pool is pinned to the CPU indexes or percentage defined in CPUAFFINITYSET, but other pools cannot use the same CPUs.
**ANY** | | ANY: Pool has no affinity for a specific CPU or percentage of available CPUs.
**CASCADETO** | VARCHAR | The name of the secondary resource pool, if one exists.

#### See Also
- CREATE RESOURCE POOL
- ALTER RESOURCE POOL

### ROLES
Contains the names of all roles the user can access, along with any roles that have been assigned to those roles.

**Tip:** You can also use the function HAS_ROLE to see if a role is available to a user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASSIGNED_ROLES</strong></td>
<td>VARCHAR</td>
<td>The names of any roles that have been granted to this role. By enabling the role, the user also has access to the privileges of these additional roles.</td>
</tr>
<tr>
<td><strong>NAME</strong></td>
<td>VARCHAR</td>
<td>The name of a role that the user can access.</td>
</tr>
<tr>
<td><strong>ROLE_ID</strong></td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the role.</td>
</tr>
</tbody>
</table>

**Note:** If you see an asterisk in the ASSIGNED_ROLES column output, it means the user has roles WITH ADMIN OPTION.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP_DN</td>
<td>VARCHAR</td>
<td>Indicates whether or not the Vertica Analytic Database role maps to an LDAP Link group. When the column is set to dn, the Vertica role maps to LDAP Link.</td>
</tr>
<tr>
<td>LDAP_URI_HASH</td>
<td>VARCHAR</td>
<td>The URI hash number for the LDAP role.</td>
</tr>
<tr>
<td>IS_ORPHANED_FROM_LDAP</td>
<td>VARCHAR</td>
<td>Indicates if the role is disconnected (orphaned) from LDAP, valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t - role is orphaned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f - role is not orphaned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information see <a href="#">Troubleshooting LDAP Link Issues</a></td>
</tr>
</tbody>
</table>

### See Also

- [GRANTS](#)
- [HAS_ROLE](#)
- [USERS](#)

### SCHEMATA

Provides information about schemas in the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the specific schema.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>Schema name for which information is listed.</td>
</tr>
<tr>
<td>SCHEMA_OWNER_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the owner who created the schema.</td>
</tr>
<tr>
<td>SCHEMA_OWNER</td>
<td>VARCHAR</td>
<td>Name of the owner who created the schema.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SYSTEM_SCHEMA_CREATOR</td>
<td>VARCHAR</td>
<td>Creator information for system schema or NULL for non-system schema.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMPTZ</td>
<td>Time when the schema was created.</td>
</tr>
<tr>
<td>IS_SYSTEM_SCHEMA</td>
<td>BOOLEAN</td>
<td>Indicates whether the schema was created for system use, where t is true and f is false.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**SEQUENCES**

Displays information about the parameters specified for a sequence using the CREATE SEQUENCE statement.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema in which the sequence was created.</td>
</tr>
<tr>
<td>SEQUENCE_NAME</td>
<td>VARCHAR</td>
<td>Name of the sequence defined in the CREATE SEQUENCE statement.</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR</td>
<td>Name of the owner.</td>
</tr>
<tr>
<td>IDENTITY_TABLE_NAME</td>
<td>VARCHAR</td>
<td>If created by an auto_increment or identity column, the name of the table to which it belongs. See column constraints in the CREATE TABLE statement.</td>
</tr>
<tr>
<td>SESSION_CACHE_COUNT</td>
<td>INTEGER</td>
<td>Count of values cached in a session.</td>
</tr>
<tr>
<td>ALLOW_CYCLE</td>
<td>BOOLEAN</td>
<td>Values allowed to cycle when a sequence reaches its minimum or maximum value. See CYCLE</td>
</tr>
<tr>
<td>OUTPUT_ORDERED</td>
<td>BOOLEAN</td>
<td>Values guaranteed to be ordered (always false).</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INCREMENT_BY</td>
<td>INTEGER</td>
<td>Sequence values are incremented by this number (negative for reverse sequences).</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>INTEGER</td>
<td>Minimum value the sequence can generate.</td>
</tr>
<tr>
<td>MAXIMUM</td>
<td>INTEGER</td>
<td>Maximum value the sequence can generate.</td>
</tr>
<tr>
<td>CURRENT_VALUE</td>
<td>INTEGER</td>
<td>Specifies how many sequence numbers Vertica has distributed to the nodes in your cluster. Includes all nodes.</td>
</tr>
<tr>
<td>SEQUENCE_SCHEMA_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the schema.</td>
</tr>
<tr>
<td>SEQUENCE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the sequence.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the user who created the sequence.</td>
</tr>
<tr>
<td>IDENTITY_TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the table to which the column belongs (if created by an auto_increment or identity column).</td>
</tr>
</tbody>
</table>

**Examples**

Create a simple sequence:

```sql
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE
```

Return information about the sequence you just created:

```sql
=> \x
Expanded display is on.
=> SELECT * FROM sequences;
- [ RECORD 1 ]------------------------------------
  sequence_schema | public
  sequence_name   | my_seq
  owner_name      | dbadmin
  identity_table_name |
  session_cache_count | 250000
  allow_cycle     | f
```
An identity column is a sequence available only for numeric column types. To identify what column in a table, if any, is an identity column, search the COLUMNS table to find the identity column in a table:

```
=> CREATE TABLE testid (c1 IDENTITY(1, 1, 1000), c2 INT);
=> \x
Expanded display is on.
=> SELECT * FROM COLUMNS WHERE is_identity='t' AND table_name='testid';
```

<table>
<thead>
<tr>
<th>table_id</th>
<th>45035996274150730</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_schema</td>
<td>public</td>
</tr>
<tr>
<td>table_name</td>
<td>testid</td>
</tr>
<tr>
<td>is_system_table</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>c1</td>
</tr>
<tr>
<td>data_type</td>
<td>int</td>
</tr>
<tr>
<td>data_type_id</td>
<td>6</td>
</tr>
<tr>
<td>data_type_length</td>
<td>8</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>1</td>
</tr>
<tr>
<td>is_nullable</td>
<td>f</td>
</tr>
<tr>
<td>column_default</td>
<td></td>
</tr>
<tr>
<td>is_identity</td>
<td>t</td>
</tr>
</tbody>
</table>

Use the SEQUENCES table to get detailed information about the sequence in testid:

```
=> SELECT * FROM sequences WHERE identity_table_name='testid';
```

<table>
<thead>
<tr>
<th>sequence_schema</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence_name</td>
<td>testid_c1_seq</td>
</tr>
<tr>
<td>owner_name</td>
<td>dbadmin</td>
</tr>
<tr>
<td>identity_table_name</td>
<td>testid</td>
</tr>
<tr>
<td>session_cache_count</td>
<td>1000</td>
</tr>
<tr>
<td>allow_cycle</td>
<td>f</td>
</tr>
<tr>
<td>output_ordered</td>
<td>f</td>
</tr>
<tr>
<td>increment_by</td>
<td>1</td>
</tr>
<tr>
<td>minimum</td>
<td>1</td>
</tr>
<tr>
<td>maximum</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>current_value</td>
<td>0</td>
</tr>
<tr>
<td>sequence_schema_id</td>
<td>45035996273704960</td>
</tr>
<tr>
<td>sequence_id</td>
<td>45035996273844996</td>
</tr>
<tr>
<td>owner_id</td>
<td>45035996273704962</td>
</tr>
<tr>
<td>identity_table_id</td>
<td>45035996274150768</td>
</tr>
</tbody>
</table>
Use the vsq1 command \ds to return a list of sequences. The following results show the two sequences created in the preceding examples. If more sequences existed, the table would list them.

The CurrentValue of the new sequence is one less than the start number you specified in the CREATE SEQUENCE and IDENTITY commands, because you have not yet used NEXTVAL to instantiate the sequences to assign their cache or supply their first start values.

```sql
=> \ds
List of Sequences
-[ RECORD 1 ]---------------------
| Schema | public |
| Sequence | my_seq |
| CurrentValue | 149 |
| IncrementBy | 1 |
| Minimum | 1 |
| Maximum | 5000 |
| AllowCycle | f |
| Comment | |
-[ RECORD 2 ]---------------------
| Schema | public |
| Sequence | testid_c1_seq |
| CurrentValue | 0 |
| IncrementBy | 1 |
| Minimum | 1 |
| Maximum | 922372036854775807 |
| AllowCycle | f |
| Comment | |
```

**STORAGE_LOCATIONS**

Provides information about storage locations, their IDs labels, and status.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the storage location.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name on which the storage location exists.</td>
</tr>
<tr>
<td>LOCATION_PATH</td>
<td>VARCHAR</td>
<td>The path where the storage location is mounted.</td>
</tr>
<tr>
<td>LOCATION_USAGE</td>
<td>VARCHAR</td>
<td>The type of information stored in the location:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DATA: Only data is stored in the location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TEMP: Only temporary files that are created during loads or queries are stored in the location.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| DATA,TEMP        |           | • DATA,TEMP: Both types of files are stored in the location.  
|                  |           | • USER: The storage location can be used by non-dbadmin users, who are granted access to the storage location  
|                  |           | • CATALOG: The area is used for the Vertica catalog. This usage is set internally and cannot be removed or changed. |
| SHARING_TYPE     | VARCHAR   | How this location is shared among database nodes, if it is:  
|                  |           | • SHARED: The path used by the storage location is used by all nodes. See the SHARED parameter to CREATE LOCATION.  
|                  |           | • COMMUNAL: the location is used for communal storage in Eon Mode Beta.  
|                  |           | • NONE: The location is not shared among nodes. |
| IS RETIRED       | BOOLEAN   | Whether the storage location has been retired. This column has a value of t (true) if the location is retired, or f (false) if it is not. |
| LOCATION_LABEL   | VARCHAR   | The label associated with a specific storage location, added with the ALTER_LOCATION_LABEL function. |
| RANK             | INTEGER   | The Access Rank value either assigned or supplied to the storage location, as described in Prioritizing Column Access Speed. |
| THROUGHPUT       | INTEGER   | The throughput performance of the storage location, measured in MB/sec. You can get location performance values using MEASURE_LOCATION_PERFORMANCE, and set them with the SET_LOCATION_PERFORMANCE function. |
| LATENCY          | INTEGER   | The measured latency of the storage location as number of data seeks per second. You can get location performance values using MEASURE_LOCATION_PERFORMANCE, and set them with the SET_LOCATION_PERFORMANCE function. |
Privileges

Must be a superuser.

See Also

- DISK_STORAGE
- MEASURE_LOCATION_PERFORMANCE
- SET_LOCATION_PERFORMANCE
- STORAGE_POLICIES
- STORAGE_USAGE
- Storage Management Functions

SYSTEM_COLUMNS

Provides table column information for SYSTEM_TABLES.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the table.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name for which information is listed.</td>
</tr>
<tr>
<td>IS_SYSTEM_TABLE</td>
<td>BOOLEAN</td>
<td>Indicates whether the table is a system table, where t is true and f is false.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>VARCHAR</td>
<td>A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The column name for which information is listed in the database.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR</td>
<td>The data type assigned to the column; for example</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
<td>VARCHARG(16).</td>
</tr>
<tr>
<td>DATA_TYPE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the data type.</td>
</tr>
<tr>
<td>DATA_TYPE_LENGTH</td>
<td>INTEGER</td>
<td>The maximum allowable length of the data type.</td>
</tr>
<tr>
<td>CHARACTER_MAXIMUM_LENGTH</td>
<td>INTEGER</td>
<td>The maximum allowable length of the column.</td>
</tr>
<tr>
<td>NUMERIC_PRECISION</td>
<td>INTEGER</td>
<td>The number of significant decimal digits.</td>
</tr>
<tr>
<td>NUMERIC_SCALE</td>
<td>INTEGER</td>
<td>The number of fractional digits.</td>
</tr>
<tr>
<td>DATETIME_PRECISION</td>
<td>INTEGER</td>
<td>For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.</td>
</tr>
<tr>
<td>INTERVAL_PRECISION</td>
<td>INTEGER</td>
<td>The number of fractional digits retained in the seconds field.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>INTEGER</td>
<td>The position of the column respective to other columns in the table.</td>
</tr>
<tr>
<td>IS_NULLABLE</td>
<td>BOOLEAN</td>
<td>Indicates whether the column can contain null values, where t is true and f is false.</td>
</tr>
<tr>
<td>COLUMN_DEFAULT</td>
<td>VARCHARG</td>
<td>The default value of a column, such as empty or expression.</td>
</tr>
</tbody>
</table>

**SYSTEM_TABLES**

Returns a list of all system table names.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the schema.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema name in which the system table resides. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• V_CATALOG</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• V_MONITOR.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the system table.</td>
</tr>
<tr>
<td>TABLE_DESCRIPTION</td>
<td>VARCHAR</td>
<td>A description of the system table's purpose.</td>
</tr>
<tr>
<td>IS_SUPERUSER_ONLY</td>
<td>BOOLEAN</td>
<td>Indicates if the table is accessible only by a user with superuser privileges. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t - table is accessible by superuser only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f - table is accessible by other users</td>
</tr>
<tr>
<td>IS_MONITORABLE</td>
<td>BOOLEAN</td>
<td>Indicates if the table is accessible by a user with the SYSMONITOR role enabled. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t - table is accessible by SYSMONITOR role</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f - table is not accessible by SYSMONITOR role</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See SYSMONITOR Role.</td>
</tr>
<tr>
<td>IS_ACCESSIBLE_DURING_LOCKDOWN</td>
<td>BOOLEAN</td>
<td>Indicates if the table is restricted. See System Table Restriction and Access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When you use the RESTRICT_SYSTEM_TABLES_ACCESS() function, the following conditions take affect:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System tables with built-in restrictions that determine who can access information remain accessible. For example, the SESSIONS table allows a user to see only information on that user's current session.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System tables that contain settings that a user should be allowed to see are accessible. For example, the TYPES system table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System tables that contain information about other users are restricted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System tables that contain information</td>
</tr>
</tbody>
</table>
The behavior of the SYSMONITOR Role does not change.

About database settings are restricted.

### TABLE_CONSTRAINTS

Provides information about table constraints.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the constraint.</td>
</tr>
<tr>
<td>CONSTRAINT_NAME</td>
<td>VARCHAR</td>
<td>The name of the constraint, if specified as UNIQUE, FOREIGN KEY, NOT NULL, PRIMARY KEY, or CHECK.</td>
</tr>
<tr>
<td>CONSTRAINT_SCHEMA_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the schema containing the constraint.</td>
</tr>
<tr>
<td>CONSTRAINT_KEY_COUNT</td>
<td>INTEGER</td>
<td>The number of constraint keys.</td>
</tr>
<tr>
<td>FOREIGN_KEY_COUNT</td>
<td>INTEGER</td>
<td>The number of foreign keys.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the table that contains the UNIQUE, FOREIGN KEY, NOT NULL, or PRIMARY KEY constraint.</td>
</tr>
<tr>
<td>FOREIGN_TABLE_ID</td>
<td>INTEGER</td>
<td>The unique object ID of the foreign table referenced in a foreign key constraint (zero if not a foreign key constraint).</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>CHAR</td>
<td>Indicates the constraint type.</td>
</tr>
<tr>
<td>Valid Values:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● c — check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● f — foreign</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLES

Provides information about all tables in the database.

**Tip:** Columns `TABLE_SCHEMA` and `TABLE_NAME` are case sensitive. To query `TABLES` on these columns, use the case-insensitive `ILIKE` predicate. For example:

```sql
SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'Store%';
```

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TABLE_SCHEMA_ID</code></td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the schema and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td><code>TABLE_SCHEMA</code></td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td><code>TABLE_ID</code></td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the table and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td><code>TABLE_NAME</code></td>
<td>VARCHAR</td>
<td>The table name for which information is listed.</td>
</tr>
<tr>
<td><code>OWNER_ID</code></td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the owner and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td><code>OWNER_NAME</code></td>
<td>VARCHAR</td>
<td>The name of the user who created the table.</td>
</tr>
<tr>
<td><code>IS_TEMP_TABLE</code></td>
<td>BOOLEAN</td>
<td>Indicates whether this table is a temporary table.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IS_SYSTEM_TABLE</td>
<td>BOOLEAN</td>
<td>Indicates whether table is a system table.</td>
</tr>
<tr>
<td>FORCE_OUTER</td>
<td>INTEGER</td>
<td>Specifies whether this table is joined to another as an inner or outer input. For details, see Controlling Join Inputs in Analyzing Data.</td>
</tr>
<tr>
<td>IS_FLEXTABLE</td>
<td>BOOLEAN</td>
<td>Indicates whether the table is a Flex table.</td>
</tr>
<tr>
<td>IS_SHARED</td>
<td>BOOLEAN</td>
<td>Indicates whether the table is located on shared storage.</td>
</tr>
<tr>
<td>HAS_AGgregate_PROJECTION</td>
<td>BOOLEAN</td>
<td>Indicates whether the table has live aggregate projections.</td>
</tr>
<tr>
<td>SYSTEM_TABLE_CREATOR</td>
<td>VARCHAR</td>
<td>The name of the process that created the table, such as Designer.</td>
</tr>
<tr>
<td>PARTITION_EXPRESSION</td>
<td>VARCHAR</td>
<td>The table's partition expression.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td>Returns the timestamp, indicating when the table was created.</td>
</tr>
<tr>
<td>TABLE_DEFINITION</td>
<td>VARCHAR</td>
<td>The COPY statement table definition. This column is applicable only to external tables.</td>
</tr>
<tr>
<td>RECOVER_PRIORITY</td>
<td>INTEGER</td>
<td>The priority rank for the table for a Recovery By Table.</td>
</tr>
<tr>
<td>STORAGE_MODE</td>
<td>INTEGER</td>
<td>Specifies the load method a table uses, set to one of the following integer values: 0: The initial value for all tables that pre-date Vertica 9.0.x 1: Direct 5: Trickle 6: Auto (default)</td>
</tr>
<tr>
<td>PARTITION_GROUP_EXPRESSION</td>
<td>VARCHAR</td>
<td>The expression of a GROUP BY clause that qualifies a table's partition clause.</td>
</tr>
</tbody>
</table>
Examples

Find when tables were created:

```sql
=> SELECT table_schema, table_name, create_time FROM tables;

<table>
<thead>
<tr>
<th>table_schema</th>
<th>table_name</th>
<th>create_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>customer_dimension</td>
<td>2011-08-15 11:18:25.784203-04</td>
</tr>
<tr>
<td>public</td>
<td>promotion_dimension</td>
<td>2011-08-15 11:18:25.850592-04</td>
</tr>
<tr>
<td>public</td>
<td>date_dimension</td>
<td>2011-08-15 11:18:25.892347-04</td>
</tr>
<tr>
<td>public</td>
<td>vendor_dimension</td>
<td>2011-08-15 11:18:25.942805-04</td>
</tr>
<tr>
<td>public</td>
<td>employee_dimension</td>
<td>2011-08-15 11:18:25.966985-04</td>
</tr>
<tr>
<td>public</td>
<td>shipping_dimension</td>
<td>2011-08-15 11:18:25.999394-04</td>
</tr>
<tr>
<td>online_sales</td>
<td>online_page_dimension</td>
<td>2011-08-15 11:18:27.007329-04</td>
</tr>
<tr>
<td>online_sales</td>
<td>call_center_dimension</td>
<td>2011-08-15 11:18:27.476844-04</td>
</tr>
<tr>
<td>online_sales</td>
<td>online_sales_fact</td>
<td>2011-08-15 11:18:27.49749-04</td>
</tr>
</tbody>
</table>

(15 rows)
```

Find out whether certain tables are temporary and flex tables:

```sql
=> SELECT distinct table_name, table_schema, is_temp_table, is_flextable FROM v_catalog.tables
   WHERE table_name ILIKE 't%';

<table>
<thead>
<tr>
<th>table_name</th>
<th>table_schema</th>
<th>is_temp_table</th>
<th>is_flextable</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2_temp</td>
<td>public</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>tt_keys</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t2_temp_keys</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t3</td>
<td>public</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>t1</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t9_keys</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t2_keys</td>
<td>public</td>
<td>f</td>
<td>t</td>
</tr>
<tr>
<td>t6</td>
<td>public</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>t5</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t2</td>
<td>public</td>
<td>f</td>
<td>t</td>
</tr>
<tr>
<td>t8</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t7</td>
<td>public</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>tt</td>
<td>public</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t2_keys_keys</td>
<td>public</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t9</td>
<td>public</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

(15 rows)
```

TEXT_INDICES

Provides summary information about the text indices in Vertica.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the index and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>INDEX_NAME</td>
<td>VARCHAR</td>
<td>The name of the text index.</td>
</tr>
<tr>
<td>INDEX_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name of the text index.</td>
</tr>
<tr>
<td>SOURCE_TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the table and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>SOURCE_TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the source table used to build the index.</td>
</tr>
<tr>
<td>SOURCE_TABLE_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name of the source table.</td>
</tr>
<tr>
<td>TOKENIZER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the tokenizer and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>TOKENIZER_NAME</td>
<td>VARCHAR</td>
<td>The name of the tokenizer used when building the index.</td>
</tr>
<tr>
<td>TOKENIZER_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name of the tokenizer.</td>
</tr>
<tr>
<td>STEMMER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the stemmer and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>STEMMER_NAME</td>
<td>VARCHAR</td>
<td>The name of the stemmer used when building the index.</td>
</tr>
<tr>
<td>STEMMER_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name of the stemmer.</td>
</tr>
<tr>
<td>TEXT_COL</td>
<td>VARCHAR</td>
<td>The text column used to build the index.</td>
</tr>
</tbody>
</table>

**TYPES**

Provides information about supported data types.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the specific data type.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ODBC_TYPE</td>
<td>INTEGER</td>
<td>The numerical ODBC type.</td>
</tr>
<tr>
<td>ODBC_SUBTYPE</td>
<td>INTEGER</td>
<td>The numerical ODBC subtype, used to differentiate types such as time and interval that have multiple subtypes.</td>
</tr>
<tr>
<td>MIN_SCALE</td>
<td>INTEGER</td>
<td>The minimum number of digits supported to the right of the decimal point for the data type.</td>
</tr>
<tr>
<td>MAX_SCALE</td>
<td>INTEGER</td>
<td>The maximum number of digits supported to the right of the decimal point for the data type. A value of 0 is used for types that do not use decimal points.</td>
</tr>
<tr>
<td>COLUMN_SIZE</td>
<td>INTEGER</td>
<td>The number of characters required to display the type. See: <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/ms711786%28v=VS.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/ms711786%28v=VS.85%29.aspx</a> for the details on COLUMN_SIZE for each type.</td>
</tr>
<tr>
<td>INTERVAL_MASK</td>
<td>INTEGER</td>
<td>For data types that are intervals, the bitmask to determine the range of the interval from the Vertica TYPE_ID. Details are available in the Vertica SDK.</td>
</tr>
<tr>
<td>TYPE_NAME</td>
<td>VARCHAR</td>
<td>The data type name associated with a particular data type ID.</td>
</tr>
<tr>
<td>CREATION_PARAMETERS</td>
<td>VARCHAR</td>
<td>A list of keywords, separated by commas, corresponding to each parameter that the application may specify in parentheses when using the name that is returned in the TYPE_NAME field. The keywords in the list can be any of the following: length, precision, or scale. They appear in the order that the syntax requires them to be used.</td>
</tr>
</tbody>
</table>

**USER_AUDITS**

Lists the results of database and object size audits generated by users calling the AUDIT function. See Monitoring Database Size for License Compliance in the Administrator's Guide for more information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE_BYTES</td>
<td>INTEGER</td>
<td>The estimated raw data size of the database</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>The ID of the user who generated the audit</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user who generated the audit</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>INTEGER</td>
<td>The ID of the object being audited</td>
</tr>
<tr>
<td>OBJECT_TYPE</td>
<td>VARCHAR</td>
<td>The type of object being audited (table, schema, etc.)</td>
</tr>
<tr>
<td>OBJECT_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema containing the object being audited</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>The name of the object being audited</td>
</tr>
<tr>
<td>AUDITED_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema on which you want to query HISTORICAL data. After running audit on a table, you can drop the table. In this case, object_schema becomes NULL.</td>
</tr>
<tr>
<td>AUDITED_OBJECT_NAME</td>
<td>VARCHAR</td>
<td>The name of the object on which you want to query HISTORICAL data. After running audit on a table, you can drop the table. In this case, object_name becomes NULL.</td>
</tr>
<tr>
<td>LICENSE_NAME</td>
<td>VARCHAR</td>
<td>The name of the license. After running a compliance audit, the value for this column is always vertica.</td>
</tr>
<tr>
<td>AUDIT_START_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>When the audit started</td>
</tr>
<tr>
<td>AUDIT_END_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>When the audit finished</td>
</tr>
<tr>
<td>CONFIDENCE_LEVEL_PERCENT</td>
<td>FLOAT</td>
<td>The confidence level of the size estimate</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
ERROR_TOLERANCE_PERCENT | FLOAT | The error tolerance used for the size estimate
USED_SAMPLING | BOOLEAN | Whether data was randomly sampled (if false, all of the data was analyzed)
CONFIDENCE_INTERVAL_LOWER_BOUND_BYTES | INTEGER | The lower bound of the data size estimate within the confidence level
CONFIDENCE_INTERVAL_UPPER_BOUND_BYTES | INTEGER | The upper bound of the data size estimate within the confidence level
SAMPLE_COUNT | INTEGER | The number of data samples used to generate the estimate
CELL_COUNT | INTEGER | The number of cells in the database

### USER_CLIENT_AUTH

Provides information about the client authentication methods that are associated with database users. You associate an authentication method with a user using `GRANT (Authentication)`.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_OID</td>
<td>INTEGER</td>
<td>A unique identifier for that user.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user.</td>
</tr>
<tr>
<td>AUTH_OID</td>
<td>INTEGER</td>
<td>A unique identifier for the authentication method you are using.</td>
</tr>
<tr>
<td>AUTH_NAME</td>
<td>VARCHAR</td>
<td>Name that you gave to the authentication method.</td>
</tr>
</tbody>
</table>
| GRANTED_TO | BOOLEAN | Name of the user with whom you have associated the authentication method using `GRANT (Authentication)`.
USER_FUNCTION_PARAMETERS

Provides information about the parameters of a C++ user-defined function (UDx). You can only view parameters that have the Properties.visible parameter set to TRUE.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR (128)</td>
<td>The schema to which the function belongs.</td>
</tr>
<tr>
<td>FUNCTION_NAME</td>
<td>VARCHAR (128)</td>
<td>The name assigned by the user to the User-Defined Function.</td>
</tr>
<tr>
<td>FUNCTION_TYPE</td>
<td>VARCHAR (128)</td>
<td>The type of user-defined function. For example, 'User Defined Function'.</td>
</tr>
<tr>
<td>FUNCTION_ARGUMENT_TYPE</td>
<td>VARCHAR (8192)</td>
<td>The number and data types of input arguments for the function.</td>
</tr>
<tr>
<td>PARAMETER_NAME</td>
<td>VARCHAR (128)</td>
<td>The name of the parameter for the user-defined function.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR (128)</td>
<td>The data type of the parameter.</td>
</tr>
<tr>
<td>DATA_TYPE_ID</td>
<td>INTEGER</td>
<td>A number specifying the ID for the parameter's data type.</td>
</tr>
<tr>
<td>DATA_TYPE_LENGTH</td>
<td>INTEGER</td>
<td>The maximum length of the parameter's data type.</td>
</tr>
<tr>
<td>IS_REQUIRED</td>
<td>BOOLEAN</td>
<td>Indicates whether the parameter is required or not.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If set to TRUE, and you don't provide the parameter, Vertica throws an error.</td>
</tr>
<tr>
<td>CAN_BE_NULL</td>
<td>BOOLEAN</td>
<td>Indicates whether the parameter can be passed as a NULL value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If set to FALSE, you pass the parameter with a NULL value, Vertica throws an error.</td>
</tr>
<tr>
<td>COMMENT</td>
<td>VARCHAR (128)</td>
<td>A user-supplied description of the parameter.</td>
</tr>
</tbody>
</table>
Privileges

Any user can query the USER_FUNCTION_PARAMETERS table. However, users can only see table information about those UDx functions which the user has permission to use.

See Also

- Developing User-Defined Extensions (UDxs)
- UDx Parameters

USER_FUNCTIONS

Returns metadata about user-defined SQL functions (which store commonly used SQL expressions as a function in the Vertica catalog) and User-Defined Functions (UDx).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema in which this function exists.</td>
</tr>
<tr>
<td>FUNCTION_NAME</td>
<td>VARCHAR</td>
<td>The name assigned by the user to the SQL function or User-Defined Function.</td>
</tr>
<tr>
<td>PROCEDURE_TYPE</td>
<td>VARCHAR</td>
<td>The type of user-defined function. For example, 'User Defined Function'.</td>
</tr>
<tr>
<td>FUNCTION_RETURN_TYPE</td>
<td>VARCHAR</td>
<td>The data type name that the SQL function returns.</td>
</tr>
<tr>
<td>FUNCTION_ARGUMENT_TYPE</td>
<td>VARCHAR</td>
<td>The number and data types of parameters for the function.</td>
</tr>
<tr>
<td>FUNCTION_DEFINITION</td>
<td>VARCHAR</td>
<td>The SQL expression that the user defined in the SQL function's function body.</td>
</tr>
<tr>
<td>VOLATILITY</td>
<td>VARCHAR</td>
<td>The SQL function's volatility (whether a function returns the same output given the same input). Can be immutable, volatile, or stable.</td>
</tr>
<tr>
<td>IS STRICT</td>
<td>BOOLEAN</td>
<td>Indicates whether the SQL function is strict, where t is true and f is false.</td>
</tr>
</tbody>
</table>
## Notes

- The volatility and strictness of a SQL function are automatically inferred from the function definition in order that Vertica determine the correctness of usage, such as where an immutable function is expected but a volatile function is provided.

- The volatility and strictness of a UDx is defined by the UDx's developer.

## Example

Create a SQL function called `myzeroifnull` in the public schema:

```sql
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
   AS BEGIN
       RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
   END;
```

Now query the USER_FUNCTIONS table. The query returns just the `myzeroifnull` macro because it is the only one created in this schema:

```sql
=> SELECT * FROM user_functions;
```

<table>
<thead>
<tr>
<th>schema_name</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_name</td>
<td>myzeroifnull</td>
</tr>
<tr>
<td>procedure_type</td>
<td>User Defined Function</td>
</tr>
<tr>
<td>function_return_type</td>
<td>Integer</td>
</tr>
<tr>
<td>function_argument_type</td>
<td>x Integer</td>
</tr>
<tr>
<td>function_definition</td>
<td>RETURN CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END</td>
</tr>
<tr>
<td>volatility</td>
<td>immutable</td>
</tr>
<tr>
<td>is_strict</td>
<td>f</td>
</tr>
<tr>
<td>is_fenced</td>
<td>f</td>
</tr>
<tr>
<td>comment</td>
<td></td>
</tr>
</tbody>
</table>
See Also

- CREATE FUNCTION (SQL Functions)
- ALTER FUNCTION (UDF)
- DROP FUNCTION

**USER_PROCEDURES**

Provides information about external procedures that have been defined for Vertica. User sees only the procedures they can execute.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEDURE_NAME</td>
<td>VARCHAR</td>
<td>The name given to the external procedure through the CREATE PROCEDURE statement.</td>
</tr>
<tr>
<td>PROCEDURE_ARGUMENTS</td>
<td>VARCHAR</td>
<td>Lists arguments for the external procedure.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>Indicates the schema in which the external procedure is defined.</td>
</tr>
</tbody>
</table>

**Example**

```sql
g=> SELECT * FROM user_procedures;
procedure_name | procedure_arguments | schema_name
---------------|---------------------|---------------------
helloplanet    | arg1 VARCHAR       | public              
(1 row)
```

**USER_TRANSFORMS**

Lists the currently-defined user-defined transform functions (UDTFs).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR (128)</td>
<td>The name of the schema containing the UDTF.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FUNCTION_NAME</td>
<td>VARCHAR(128)</td>
<td>The SQL function name assigned by the user.</td>
</tr>
<tr>
<td>FUNCTION_RETURN_TYPE</td>
<td>VARCHAR(128)</td>
<td>The data types of the columns the UDTF returns.</td>
</tr>
<tr>
<td>FUNCTION_ARGUMENT_TYPE</td>
<td>VARCHAR(8192)</td>
<td>The data types of the columns that make up the input row.</td>
</tr>
<tr>
<td>FUNCTION_DEFINITION</td>
<td>VARCHAR(128)</td>
<td>A string containing the name of the factory class for the UDTF, and the name of the library that contains it.</td>
</tr>
<tr>
<td>IS_FENCED</td>
<td>BOOLEAN</td>
<td>Whether the UDTF runs in fenced mode.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit permissions are required; however, users see only UDTFs contained in schemas to which they have read access.

**See Also**

- Transform Functions (UDTFs)
- CREATE TRANSFORM FUNCTION

**USERS**

Provides information about all users in the database.

**Tip**: To see if a role has been assigned to a user, call the function `HAS_ROLE`.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the user.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The user name for which information is listed.</td>
</tr>
<tr>
<td>IS_SUPER_USER</td>
<td>BOOLEAN</td>
<td>A system flag, where $t$ (true) identifies the superuser created at the time of installation. All other users are denoted by $f$ (false).</td>
</tr>
<tr>
<td>PROFILE_NAME</td>
<td>VARCHAR</td>
<td>The name of the profile to which the user is assigned. The profile controls the user's password policy.</td>
</tr>
<tr>
<td>IS_LOCKED</td>
<td>BOOLEAN</td>
<td>Whether the user's account is locked. A locked user cannot log into the system.</td>
</tr>
<tr>
<td>LOCK_TIME</td>
<td>TIMESTAMPTZ</td>
<td>When the user's account was locked. Used to determine when to automatically unlock the account, if the user's profile has a PASSWORD_LOCK_TIME parameter set.</td>
</tr>
<tr>
<td>RESOURCE_POOL</td>
<td>VARCHAR</td>
<td>The resource pool to which the user is assigned.</td>
</tr>
<tr>
<td>MEMORY_CAP_KB</td>
<td>VARCHAR</td>
<td>The maximum amount of memory a query run by the user can consume, in kilobytes.</td>
</tr>
<tr>
<td>TEMP_SPACE_CAP_KB</td>
<td>VARCHAR</td>
<td>The maximum amount of temporary disk space a query run by the user can consume, in kilobytes.</td>
</tr>
<tr>
<td>RUN_TIME_CAP</td>
<td>VARCHAR</td>
<td>The maximum amount of time any of the user's queries are allowed to run.</td>
</tr>
<tr>
<td>MAX_CONNECTIONS</td>
<td>VARCHAR</td>
<td>The maximum number of connections allowed for this user.</td>
</tr>
<tr>
<td>CONNECTION_LIMIT_MODE</td>
<td>VARCHAR</td>
<td>Indicates whether the user sets connection limits through the node or in database mode.</td>
</tr>
<tr>
<td>IDLE_SESSION_TIMEOUT</td>
<td>VARCHAR</td>
<td>The time the system waits before timing out the user's idle session. Maximum value is 1 year. See Interval Subtype Units for valid intervals.</td>
</tr>
<tr>
<td>GRACE_PERIOD</td>
<td>VARCHAR</td>
<td>Specifies how long a user query can block on any session socket, while awaiting client input or output. If the socket is blocked for a continuous period that</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated.</td>
</tr>
<tr>
<td>ALL_ROLES</td>
<td>VARCHAR</td>
<td>Roles assigned to the user. An asterisk in ALL.Roles output means role granted WITH ADMIN OPTION. See Database Roles in the Administrator's Guide.</td>
</tr>
<tr>
<td>DEFAULT_ROLES</td>
<td>VARCHAR</td>
<td>Default roles assigned to the user. An asterisk in DEFAULT_ROLES output means role granted WITH ADMIN OPTION. See Default Roles for Database Users in the Administrator's Guide.</td>
</tr>
<tr>
<td>SEARCH_PATH</td>
<td>VARCHAR</td>
<td>Sets the default schema search path for the user. See Setting Search Paths in the Administrator's Guide.</td>
</tr>
<tr>
<td>LDAP_DN</td>
<td>VARCHAR</td>
<td>Indicates whether or not the Vertica Analytic Database user maps to an LDAP Link user. When the column is set to dn, the Vertica user maps to LDAP Link..</td>
</tr>
<tr>
<td>LDAP_URI_HASH</td>
<td>INTEGER</td>
<td>The URI hash number for the LDAP user.</td>
</tr>
<tr>
<td>IS_ORPHANED_FROM_LDAP</td>
<td>BOOLEAN</td>
<td>Indicates if the user is disconnected (orphaned) from LDAP, set to one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• t: User is orphaned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• f: User is not orphaned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information see Troubleshooting LDAP Link Issues</td>
</tr>
</tbody>
</table>

See Also

- GRANTS
- HAS_ROLE
VIEW_COLUMNS

Provides view attribute information.

Note: If you drop a table that is referenced by a view, Vertica does not drop the view. However, attempts to access information about it from VIEW_COLUMNS return an error that the view is invalid.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies this view.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of this view's schema.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The view name.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>VARCHAR</td>
<td>A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in this view.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The name of a column in this view.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>VARCHAR</td>
<td>The data type of a view column.</td>
</tr>
<tr>
<td>DATA_TYPE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies a view column's data type.</td>
</tr>
<tr>
<td>DATA_TYPE_LENGTH</td>
<td>INTEGER</td>
<td>The data type's maximum length.</td>
</tr>
<tr>
<td>CHARACTER_MAXIMUM_LENGTH</td>
<td>INTEGER</td>
<td>The column's maximum length, valid only for character types.</td>
</tr>
<tr>
<td>NUMERIC_PRECISION</td>
<td>INTEGER</td>
<td>The column's number of significant decimal digits.</td>
</tr>
<tr>
<td>NUMERIC_SCALE</td>
<td>INTEGER</td>
<td>The column's number of fractional digits.</td>
</tr>
<tr>
<td>DATETIME_PRECISION</td>
<td>INTEGER</td>
<td>For TIMESTAMP data type, returns the declared precision; returns null if no precision was declared.</td>
</tr>
<tr>
<td>INTERVAL_PRECISION</td>
<td>INTEGER</td>
<td>The number of fractional digits retained in the seconds field.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>INTEGER</td>
<td>The position of the column relative to other columns in the view.</td>
</tr>
</tbody>
</table>

See Also

**VIEWS**

Provides information about all views within the system. See Views for more information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the view schema.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the view schema.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the view.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The view name.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the view owner.</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR</td>
<td>View owner's user name</td>
</tr>
<tr>
<td>VIEW_DEFINITION</td>
<td>VARCHAR</td>
<td>The query that defines the view.</td>
</tr>
<tr>
<td>IS_SYSTEM_VIEW</td>
<td>BOOLEAN</td>
<td>Indicates whether the view is a system view.</td>
</tr>
<tr>
<td>SYSTEM_VIEW_CREATOR</td>
<td>VARCHAR</td>
<td>View creator's user name</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td>Specifies when this view was created.</td>
</tr>
<tr>
<td>IS_LOCAL_TEMP_VIEW</td>
<td>BOOLEAN</td>
<td>Indicates whether this view is a temporary view stored locally.</td>
</tr>
<tr>
<td>INHERIT_</td>
<td>BOOLEAN</td>
<td>Indicates whether inherited privileges are enabled</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>PRIVILEGES</td>
<td></td>
<td>for this view. For details, see Grant Inherited Privileges.</td>
</tr>
</tbody>
</table>

See Also

VIEW_COLUMNS
V_MONITOR Schema

The system tables in this section reside in the v_monitor schema. These tables provide information about the health of the Vertica database.

ACTIVE_EVENTS

Returns all active events in the cluster. See Monitoring Events.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name where the event occurred.</td>
</tr>
<tr>
<td>EVENT_CODE</td>
<td>INTEGER</td>
<td>A numeric ID that indicates the type of event. See Event Types for a list of event type codes.</td>
</tr>
<tr>
<td>EVENT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the specific event.</td>
</tr>
<tr>
<td>EVENT_SEVERITY</td>
<td>VARCHAR</td>
<td>The severity of the event from highest to lowest. These events are based on standard syslog severity types.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0—Emergency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1—Alert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2—Critical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3—Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 4—Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 5—Notice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 6—Informational</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 7—Debug</td>
</tr>
<tr>
<td>EVENT_POSTED_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The year, month, day, and time the event was reported. The time is posted in military time.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EVENT_EXPIRATION</td>
<td>VARCHAR</td>
<td>The year, month, day, and time the event expire. The time is posted in military time. If the cause of the event is still active, the event is posted again.</td>
</tr>
<tr>
<td>EVENT_CODE_DESCRIPTION</td>
<td>VARCHAR</td>
<td>A brief description of the event and details pertinent to the specific situation.</td>
</tr>
<tr>
<td>EVENT_PROBLEM_DESCRIPTION</td>
<td>VARCHAR</td>
<td>A generic description of the event.</td>
</tr>
<tr>
<td>REPORTING_NODE</td>
<td>VARCHAR</td>
<td>The name of the node within the cluster that reported the event.</td>
</tr>
<tr>
<td>EVENT_SENT_TO_CHANNELS</td>
<td>VARCHAR</td>
<td>The event logging mechanisms that are configured for Vertica. These can include vertica.log, (configured by default) syslog, and SNMP.</td>
</tr>
<tr>
<td>EVENT_POSTED_COUNT</td>
<td>INTEGER</td>
<td>Tracks the number of times an event occurs. Rather than posting the same event multiple times, Vertica posts the event once and then counts the number of additional instances in which the event occurs.</td>
</tr>
</tbody>
</table>

**ALLOCATOR_USAGE**

Provides real-time information on the allocation and reuse of memory pools for a Vertica node.

There are two memory pools in Vertica, global and SAL. The global memory pool is related to Vertica catalog objects. The SAL memory pool is related to the system storage layer. These memory pools are physical structures from which Vertica allocates and reuses portions of memory.

Within the memory pools, there are two allocation types. Both global and SAL memory pools include chunk and object memory allocation types.

- *Chunk* allocations are from tiered storage, and are grouped into sizes, in bytes, that are powers of 2.
- *Object* allocations are object types, for example, a table or projection. Each object assumes a set size.
The table provides detailed information on these memory pool allocations.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node from which Vertica has collected this allocator information.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>One of two memory pools:</td>
</tr>
<tr>
<td>ALLOCATION_TYPE</td>
<td>VARCHAR</td>
<td>One of two memory allocation types:</td>
</tr>
<tr>
<td>UNIT_SIZE</td>
<td>INTEGER</td>
<td>The size, in bytes, of the memory allocation. For example, if the allocation type is a table (an object type), then Vertica allots 8 bytes.</td>
</tr>
<tr>
<td>FREE_COUNT</td>
<td>INTEGER</td>
<td>Indicates the count of blocks of freed memory that Vertica has reserved for future memory needs. For example, if you delete a table, Vertica reserves the 8 bytes originally allotted for the table. The 8 bytes freed become 1 unit of memory that Vertica adds to this column.</td>
</tr>
<tr>
<td>FREE_BYTES</td>
<td>INTEGER</td>
<td>Indicates the number of freed memory bytes. For example, with a table deletion, Vertica adds 8 bytes to this column.</td>
</tr>
<tr>
<td>USED_COUNT</td>
<td>INTEGER</td>
<td>Indicates the count of in-use blocks for this allocation.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, if your database includes two table objects, Vertica adds 2 to this column.</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes of in-use blocks of memory. For example, if your database includes two table objects, each of which assume 8 bytes, Vertica adds 16 to this column.</td>
</tr>
<tr>
<td>TOTAL_SIZE</td>
<td>INTEGER</td>
<td>Indicates the number of bytes that is the sum of all free and used memory.</td>
</tr>
<tr>
<td>CAPTURE_TIME</td>
<td>TIMESTAMPTZ</td>
<td>Indicates the current timestamp for when Vertica collected the for this table.</td>
</tr>
<tr>
<td>ALLOCATION_NAME</td>
<td>VARCHAR</td>
<td>Provides the name of the allocation type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the allocation is an object type, provides the name of the object. For example, CAT::Schema. Object types can also have the name internal, meaning that the object is an internal data structure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Those object types that are not internal are prefaced with either CAT or SAL. Those prefaced with CAT indicate memory from the global memory pool. SAL indicates memory from the system storage memory pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the allocation type is chunk, indicates a power of 2 in this field to represent the number of bytes assumed by the chunk. For example, $2^5$.</td>
</tr>
</tbody>
</table>

**Sample: How Memory Pool Memory is Allotted, Retained, and Freed**

The following table shows sample column values based upon a hypothetical example. The sample illustrates how column values change based upon addition or deletion of a table object.
When you add a table object (t1), Vertica assumes a UNIT_SIZE of 8 bytes, with a USED_COUNT of 1.

When you add a second table object (t2), the USED_COUNT increases to 2. Since each object assumes 8 bytes, USED_BYTES increases to 16.

When you delete one of the two table objects, Vertica USED_COUNT decreases to 1, and USED_BYTES decreases to 8. Since Vertica retains the memory for future use, FREE_BYTES increases to 8, and FREE_COUNT increases to 1.

Finally, when you create a new table object (t3), Vertica frees the memory for reuse. FREE_COUNT and FREE_BYTES return to 0.

<table>
<thead>
<tr>
<th>Column Names</th>
<th>Add One Table Object (t1)</th>
<th>Add a Second Table Object (t2)</th>
<th>Delete a Table Object (t2)</th>
<th>Create a New Table Object (t3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>v_vmart_node0001</td>
<td>v_vmart_node0001</td>
<td>v_vmart_node0001</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>global</td>
<td>global</td>
<td>global</td>
<td>global</td>
</tr>
<tr>
<td>ALLOCATION_TYPE</td>
<td>object</td>
<td>object</td>
<td>object</td>
<td>object</td>
</tr>
<tr>
<td>UNIT_SIZE</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>FREE_COUNT</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>FREE_BYTES</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>USED_COUNT</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>TOTAL_SIZE</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>ALLOCATION_NAME</td>
<td>CAT::Table</td>
<td>CAT::Table</td>
<td>CAT::Table</td>
<td>CAT::Table</td>
</tr>
</tbody>
</table>
Example

The following example shows one sample record for a chunk allocation type, and one for an object type.

```sql
=> \x
Expanded display is on.

=> select * from allocator_usage;
-[
  RECORD 1 ]-----------------------------
node_name          | v_vmart_node0004
pool_name          | global
allocation_type    | chunk
unit_size          | 8
free_count         | 1869
free_bytes         | 8552
used_count         | 7327
used_bytes         | 58616
total_size         | 67168
allocation_name    | 2^3

[
  RECORD 105 ]-----------------------------
node_name          | v_vmart_node0004
pool_name          | SAL
allocation_type    | object
unit_size          | 128
free_count         | 0
free_bytes         | 0
used_count         | 2
used_bytes         | 256
total_size         | 256
capture_time       | 2017-05-24 14:44:30.153892-04
allocation_name    | SAL::WOSAlloc

```

CATALOG_SUBSCRIPTION_CHANGES

Lists the changes made to catalog subscriptions.
### Column Name | Data Type | Description
---|---|---
EVENT_TIMESTAMP | TIMESTAMP | The time a catalog subscription changed.
SESSION_ID | VARCHAR | A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
USER_ID | INTEGER | A unique numeric ID assigned by the Vertica catalog, which identifies the user.
USER_NAME | VARCHAR | The user who made changes to the subscriptions.
TRANSACTION_ID | INTEGER | Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.
SHARD_NAME | VARCHAR | The name of the shard.
SHARD_OID | INTEGER | The OID of the shard.
SUBSCRIBER_NODE_NAME | VARCHAR | The node name or names subscribed to the shard.
SUBSCRIBER_NODE_OID | INTEGER | The OID of the subscribing node or nodes.
OLD_STATE | VARCHAR | The previous state of the node subscription.
NEW_STATE | VARCHAR | The current state of the node subscription.
WAS_PRIMARY | BOOLEAN | Defines whether the node was the primary subscriber.
IS_PRIMARY | BOOLEAN | Defines whether the node is currently the primary subscriber.
CATALOG_VERSION | INTEGER | The version of the catalog at the time of the subscription change.

**CATALOG_SYNC_STATE**

Indicates the last time a node synchronized its catalog.
### Column Name | Data Type | Description
--- | --- | ---
**NODE_OID** | INTEGER | The OID of the node.
**SYNC_CATALOG_VERSION** | INTEGER | The version number of the catalog being synchronized.
**LAST_CONFIG_CHANGE_VERSION** | INTEGER | The version number of the catalog before the next synchronization.
**SYNC_TRAILING_INTERVAL** | INTEGER | The difference between the global catalog version and the synchronized catalog version for a node.
**LAST_SYNC_AT** | DATE/TIME | The date and time the last time the catalog was synchronized.
**SYNC_QUEUE_SIZE** | INTEGER | The size, in bytes, of the synchronization queue.

### Example

```sql
=> SELECT * FROM CATALOG_SYNC_STATE;
```

<table>
<thead>
<tr>
<th>node_oid</th>
<th>45035996273704980</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync_catalog_version</td>
<td>99</td>
</tr>
<tr>
<td>last_config_change_version</td>
<td>99</td>
</tr>
<tr>
<td>sync_trailing_interval</td>
<td>0</td>
</tr>
<tr>
<td>last_sync_at</td>
<td>2017-10-02 00:01:08.012556-04</td>
</tr>
<tr>
<td>sync_queue_size</td>
<td>0</td>
</tr>
</tbody>
</table>

### CATALOG_TRUNCATION_STATUS

Indicates how far behind a catalog version may be after you revive a cluster.

### Column Name | Data Type | Description
--- | --- | ---
**NODE_OID** | INTEGER | The OID of the node.
**CURRENT_CATALOG_VERSION** | INTEGER | The version number of the catalog currently on the cluster.
**TRUNCATION_CATALOG_VERSION** | INTEGER | The version number of the catalog after the next synchronization.
Example

```sql
=> SELECT * FROM CATALOG_TRUNCATION_STATUS;
- [ RECORD 1 ]---------------------------------------
node_oid | 45035996273704980
current_catalog_version | 99
truncation_catalog_version | 99
```

### COLUMN_STORAGE

Returns the amount of disk storage used by each column of each projection on each node. WOS data is stored by row, so per-column byte counts are not available.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the column.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The column name for which information is listed.</td>
</tr>
<tr>
<td>ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in the column.</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>INTEGER</td>
<td>The disk storage allocation of the column in bytes.</td>
</tr>
<tr>
<td>ENCODINGS</td>
<td>VARCHAR</td>
<td>The encoding type for the column.</td>
</tr>
<tr>
<td>COMPRESSION</td>
<td>VARCHAR</td>
<td>The compression type for the column. You can compare ENCODINGS and COMPRESSION columns to see how different encoding types affect column storage when optimizing for compression.</td>
</tr>
<tr>
<td>WOS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of WOS rows in the column.</td>
</tr>
<tr>
<td>ROS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of ROS rows in the column.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ROS_USED_BYTES</td>
<td>INTEGER</td>
<td>The number of ROS bytes in the column.</td>
</tr>
<tr>
<td>ROS_COUNT</td>
<td>INTEGER</td>
<td>The number of ROS containers.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The associated projection name for the column.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema associated with the projection.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the anchor table.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>The associated table name.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The associated table's schema name.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_COLUMN_ID</td>
<td>VARCHAR</td>
<td>A unique VARCHAR ID, assigned by the Vertica catalog, that identifies a column in a table.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_COLUMN_NAME</td>
<td>VARCHAR</td>
<td>The name of the anchor table.</td>
</tr>
</tbody>
</table>

**CONFIGURATION_CHANGE**

Records the change history of system configuration parameters. This information is useful for identifying:

- Who changed the configuration parameter value
- When the configuration parameter was changed
- Whether nonstandard settings were in effect in the past

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when the row was recorded.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Identifier of the user who changed configuration parameters.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user who changed configuration parameters at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>VARCHAR</td>
<td>Name of the changed parameter. See Configuration Parameters in the Administrator's Guide for a detailed list of supported parameters.</td>
</tr>
<tr>
<td>VALUE</td>
<td>VARCHAR</td>
<td>New value of the configuration parameter.</td>
</tr>
</tbody>
</table>

### Privileges

Superuser

### CONFIGURATION_PARAMETERS

Provides information about configuration parameters currently in use by the system that are configurable at the database, node, or session level.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node names on the cluster for which information is listed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALL indicates that all the nodes have the same value.</td>
</tr>
<tr>
<td>PARAMETER_NAME</td>
<td>VARCHAR</td>
<td>The name of the configurable parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For names of supported parameters, see Configuration Parameter Categories in the Administrator's Guide.</td>
</tr>
<tr>
<td>CURRENT_VALUE</td>
<td>VARCHAR</td>
<td>The value of the current setting for the</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RESTART_VALUE</td>
<td>VARCHAR</td>
<td>The value of the parameter after the next restart.</td>
</tr>
<tr>
<td>DATABASE_VALUE</td>
<td>VARCHAR</td>
<td>The value that is set at the database level. If no database-level value is set, the value reflects the default value.</td>
</tr>
<tr>
<td>DEFAULT_VALUE</td>
<td>VARCHAR</td>
<td>The default value for the parameter.</td>
</tr>
<tr>
<td>CURRENT_LEVEL</td>
<td>VARCHAR</td>
<td>Level at which CURRENT_VALUE is set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong> Node, database, session, or default.</td>
</tr>
<tr>
<td>RESTART_LEVEL</td>
<td>VARCHAR</td>
<td>Level at which the parameter will be set after the next restart.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong> Node, database, or default.</td>
</tr>
<tr>
<td>IS_MISMATCH</td>
<td>BOOLEAN</td>
<td>A <code>t</code> (true) setting indicates CURRENT_VALUE and RESTART_VALUE do not match.</td>
</tr>
<tr>
<td>GROUPS</td>
<td>VARCHAR</td>
<td>Any group to which the parameter belongs (for example, Security Parameters).</td>
</tr>
<tr>
<td>ALLOWED_LEVELS</td>
<td>VARCHAR</td>
<td>Indicates level or levels at which the specified parameter can be set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong> Node, database, or session.</td>
</tr>
<tr>
<td>SUPERUSER_ONLY</td>
<td>BOOLEAN</td>
<td>Indicates if the parameter settings are viewable by the superuser only. If true, the following columns will be masked if viewed by a non-superuser:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• current_value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• restart_value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• database_value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• default_value</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHANGE_UNDER_SUPPORT_GUIDANCE</td>
<td>BOOLEAN</td>
<td>A t (true) setting indicates parameters intended for use only by Vertica.</td>
</tr>
<tr>
<td>CHANGEQUIRES_RESTART</td>
<td>BOOLEAN</td>
<td>Indicates whether the configuration change requires a restart, where t is true and f is false.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>A description of the parameter's purpose.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows a case where the parameter requires a restart for the new setting to take effect:

```sql
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'EnableSSL';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_name</td>
<td>EnableSSL</td>
</tr>
<tr>
<td>current_value</td>
<td>0</td>
</tr>
<tr>
<td>restart_value</td>
<td>1</td>
</tr>
<tr>
<td>database_value</td>
<td>0</td>
</tr>
<tr>
<td>default_value</td>
<td>0</td>
</tr>
<tr>
<td>current_level</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>restart_level</td>
<td>NODE</td>
</tr>
<tr>
<td>is_mismatch</td>
<td>t</td>
</tr>
<tr>
<td>groups</td>
<td></td>
</tr>
<tr>
<td>allowed_levels</td>
<td>NODE, DATABASE</td>
</tr>
<tr>
<td>superuser_only</td>
<td>f</td>
</tr>
<tr>
<td>change_under_support_guidance</td>
<td>f</td>
</tr>
<tr>
<td>change_requires_restart</td>
<td>t</td>
</tr>
<tr>
<td>description</td>
<td>Enable SSL for the server</td>
</tr>
</tbody>
</table>

The following example shows a case where a non-superuser is viewing a parameter where superuser_only is true.

```sql
=> \c VMart nonSuperuser
You are now connected to database "VMart" as user "nonSuperuser".
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE where superuser_only = 't';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_name</td>
<td>LDAPLinkDryRun</td>
</tr>
<tr>
<td>current_value</td>
<td>******</td>
</tr>
<tr>
<td>restart_value</td>
<td>******</td>
</tr>
<tr>
<td>database_value</td>
<td>******</td>
</tr>
<tr>
<td>default_value</td>
<td>******</td>
</tr>
<tr>
<td>current_level</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>restart_level</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>
See Also

Configuration Parameters in the Administrator's Guide

CPU_USAGE

Records CPU usage history on the system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Beginning of history interval.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>End of history interval.</td>
</tr>
<tr>
<td>AVERAGE_CPU_USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Average CPU usage in percent of total CPU time (0-100) during history interval.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

CRITICAL_HOSTS

Lists the critical hosts whose failure would cause the database to become unsafe and force a shutdown.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_NAME</td>
<td>VARCHAR</td>
<td>Name of a critical host</td>
</tr>
</tbody>
</table>
Privileges

None

CRITICAL_NODES

Lists the critical nodes whose failure would cause the database to become unsafe and force a shutdown.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the node.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of a critical node.</td>
</tr>
</tbody>
</table>

CURRENT_SESSION

Returns information about the current active session. Use this table to find out the current session's sessionID and get the duration of the previously-run query.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node for which information is listed</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name used to log into the database, NULL if the session is internal</td>
</tr>
<tr>
<td>CLIENT_HOSTNAME</td>
<td>VARCHAR</td>
<td>Host name and port of the TCP socket from which the client connection was made, NULL if the session is internal</td>
</tr>
<tr>
<td>TYPE</td>
<td>INTEGER</td>
<td>Identifies the session type, one of the following integer values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Client 8: Shutdown</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 DBD License audit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 Merge out Timer service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Move out Connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 Rebalance cluster VSpread</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 Recovery Sub-session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 Refresh Repartition table</td>
</tr>
<tr>
<td>CLIENT_PID</td>
<td>INTEGER</td>
<td>Process identifier of the client process that issued this connection. This process might be on a different machine than the server.</td>
</tr>
<tr>
<td>LOGIN_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>When the user logged into the database or the internal session was created. This column can help identify open sessions that are idle.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time, but can be reused when the session closes.</td>
</tr>
<tr>
<td>CLIENT_LABEL</td>
<td>VARCHAR</td>
<td>User-specified label for the client connection that can be set when using ODBC. See Label in Data Source Name (DSN) Connection Properties in Connecting to Vertica.</td>
</tr>
<tr>
<td>TRANSACTION_START</td>
<td>TIMESTAMP</td>
<td>When the current transaction started, NULL if no transaction is running.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>VARCHAR</td>
<td>Hexadecimal identifier of the current transaction, NULL if no transaction is in progress.</td>
</tr>
</tbody>
</table>
| TRANSACTION_DESCRIPTION | VARCHAR | Description of the current transaction.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT_START</td>
<td>TIMESTAMP</td>
<td>When the current statement started execution, NULL if no statement is running</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>VARCHAR</td>
<td>Unique numeric ID for the currently-running statement, NULL if no statement is being processed. Combined, TRANSACTION_ID and STATEMENT_ID uniquely identify a statement within a session.</td>
</tr>
<tr>
<td>LAST_STATEMENT_DURATION_US</td>
<td>INTEGER</td>
<td>Duration in microseconds of the last completed statement</td>
</tr>
<tr>
<td>CURRENT_STATEMENT</td>
<td>VARCHAR</td>
<td>The currently-running statement, if any. NULL indicates that no statement is currently being processed.</td>
</tr>
<tr>
<td>LAST_STATEMENT</td>
<td>VARCHAR</td>
<td>NULL if the user has just logged in, otherwise the currently running statement or most recently completed statement.</td>
</tr>
<tr>
<td>EXECUTION_ENGINE_PROFLING_CONFIGURATION</td>
<td>VARCHAR</td>
<td>See Profiling Settings below.</td>
</tr>
<tr>
<td>QUERY_PROFLING_CONFIGURATION</td>
<td>VARCHAR</td>
<td>See Profiling Settings below.</td>
</tr>
<tr>
<td>SESSION_PROFLING_CONFIGURATION</td>
<td>VARCHAR</td>
<td>See Profiling Settings below.</td>
</tr>
<tr>
<td>CLIENT_TYPE</td>
<td>VARCHAR</td>
<td>Type of client from which the connection was made, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ADO.NET Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ODBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JDBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• vsql</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td>Client version</td>
</tr>
</tbody>
</table>

Profiling Settings:
- EXECUTION_ENGINE_PROFLING_CONFIGURATION
- QUERY_PROFLING_CONFIGURATION
- SESSION_PROFLING_CONFIGURATION
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_OS</td>
<td>VARCHAR</td>
<td>Client operating system</td>
</tr>
<tr>
<td>CLIENT_OS_USER_NAME</td>
<td>VARCHAR</td>
<td>Identifies the user that logged into the database, also set for unsuccessful login attempts.</td>
</tr>
<tr>
<td>REQUESTED_PROTOCOL</td>
<td>VARCHAR</td>
<td>Communication protocol version that the ODBC client driver sends to Vertica server, used to support backward compatibility with earlier server versions.</td>
</tr>
<tr>
<td>EFFECTIVE_PROTOCOL</td>
<td>VARCHAR</td>
<td>Minimum protocol version supported by client and driver.</td>
</tr>
</tbody>
</table>

### Profiling Settings

The following columns show settings for different profiling categories:

- EXECUTIONENGINE_PROFILING_CONFIGURATION
- QUERY_PROFILING_CONFIGURATION
- SESSION_PROFILING_CONFIGURATION

These are set to one of the following:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>No profiling is set</td>
</tr>
<tr>
<td>Session</td>
<td>On for current session</td>
</tr>
<tr>
<td>Global</td>
<td>On by default for all sessions</td>
</tr>
<tr>
<td>Session, Global</td>
<td>On by default for all sessions, including current session.</td>
</tr>
</tbody>
</table>

For information about controlling profiling settings, see Enabling and Disabling Profiling in the Administrator's Guide.

### DATA_COLLECTOR

Shows the Data Collector components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. DATA_
COLLECTOR also calculates approximate collection rate, to aid in sizing calculations.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name on which information is retained.</td>
</tr>
<tr>
<td>COMPONENT</td>
<td>VARCHAR</td>
<td>The name of the component and its policy.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The data collector (dc) table name for which information is listed.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>A short description about the component.</td>
</tr>
<tr>
<td>IN_DB_LOG</td>
<td>BOOLEAN</td>
<td>Denotes if monitoring information is retained in the dbLog file.</td>
</tr>
<tr>
<td>IN_VERTICA_LOG</td>
<td>BOOLEAN</td>
<td>Denotes if monitoring information is retained in the vertica.log file.</td>
</tr>
<tr>
<td>MEMORY_BUFFER_SIZE_KB</td>
<td>INTEGER</td>
<td>The size of the memory buffer in kilobytes.</td>
</tr>
<tr>
<td>DISK_SIZE_KB</td>
<td>INTEGER</td>
<td>The on-disk size of the table in kilobytes.</td>
</tr>
<tr>
<td>SET_INTERVAL</td>
<td>BOOLEAN</td>
<td>A t (true) setting indicates time-based retention is set.</td>
</tr>
<tr>
<td>INTERVAL_TIME_S</td>
<td>INTERVAL</td>
<td>The time of retention expressed as an INTERVAL type. To turn time-based retention off, set the value to 0.</td>
</tr>
<tr>
<td>RECORD_TOO_BIG_ERRORS</td>
<td>INTEGER</td>
<td>A number that increments by one each time an error is thrown because data did not fit in memory (based on the data collector retention policy).</td>
</tr>
<tr>
<td>LOST_BUFFERS</td>
<td>INTEGER</td>
<td>The number of buffers lost.</td>
</tr>
<tr>
<td>LOST_RECORDS</td>
<td>INTEGER</td>
<td>The number of records lost.</td>
</tr>
<tr>
<td>RETIRED_FILES</td>
<td>INTEGER</td>
<td>The number of retired files.</td>
</tr>
<tr>
<td>RETIRED_RECORDS</td>
<td>INTEGER</td>
<td>The number of retired records.</td>
</tr>
<tr>
<td>CURRENT_MEMORY_RECORDS</td>
<td>INTEGER</td>
<td>The current number of rows in memory.</td>
</tr>
<tr>
<td>CURRENT_DISK_RECORDS</td>
<td>INTEGER</td>
<td>The current number of rows stored on disk.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>CURRENT_MEMORY_BYTES</td>
<td>INTEGER</td>
<td>Total current memory used in kilobytes.</td>
</tr>
<tr>
<td>CURRENT_DISK_BYTES</td>
<td>INTEGER</td>
<td>Total current disk space used in kilobytes.</td>
</tr>
<tr>
<td>FIRST_TIME</td>
<td>TIMESTAMP</td>
<td>Timestamp of the first record.</td>
</tr>
<tr>
<td>LAST_TIME</td>
<td>_TIMESTAMP</td>
<td>Timestamp of the last record</td>
</tr>
<tr>
<td>KB_PER_DAY</td>
<td>FLOAT</td>
<td>Total kilobytes used per day.</td>
</tr>
</tbody>
</table>

**Notes**

- Data Collector is on by default. To turn it off, see [Enabling and Disabling Data Collector](#).
- You can configure monitoring information retention policies. See [Data Collector Functions](#) and [Configuring Data Retention Policies](#) for more information.

**Examples**

The following example shows how to return all component names and their descriptions. This is a useful query if you want to change the retention policy for a particular component and don't remember its name:

```sql
=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;
```

<table>
<thead>
<tr>
<th>component</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllocationPoolStatistics</td>
<td>Information about global memory pools ...</td>
</tr>
<tr>
<td>AllocationPoolStatisticsByDay</td>
<td>Information about global memory pools, ... (historical, by day)</td>
</tr>
<tr>
<td>AllocationPoolStatisticsByHour</td>
<td>Information about global memory pools, ... (historical, by hour)</td>
</tr>
<tr>
<td>AllocationPoolStatisticsByMinute</td>
<td>Information about global memory pools, ... (historical, by minute)</td>
</tr>
<tr>
<td>AllocationPoolStatisticsBySecond</td>
<td>Information about global memory pools, ... (historical, by second)</td>
</tr>
<tr>
<td>AnalyzeStatistics</td>
<td>History of statistics collection</td>
</tr>
<tr>
<td>Backups</td>
<td>Monitoring successful backups</td>
</tr>
<tr>
<td>CatalogInfo</td>
<td>Catalog statistics and history</td>
</tr>
<tr>
<td>CatalogInfoByDay</td>
<td>Catalog statistics and history (historical, by day)</td>
</tr>
<tr>
<td>CatalogInfoByHour</td>
<td>Catalog statistics and history (historical, by hour)</td>
</tr>
<tr>
<td>CatalogInfoByMinute</td>
<td>Catalog statistics and history (historical, by minute)</td>
</tr>
<tr>
<td>CatalogInfoBySecond</td>
<td>Catalog statistics and history (historical, by second)</td>
</tr>
<tr>
<td>ClientServerMessages</td>
<td>Client-Server Messages (Front End to Back End Protocol) sent</td>
</tr>
<tr>
<td>ConfigurationChanges</td>
<td>Changes to configuration parameters (vertica.conf)</td>
</tr>
<tr>
<td>CpuAggregate</td>
<td>Aggregate CPU information</td>
</tr>
<tr>
<td>CpuAggregateByDay</td>
<td>Aggregate CPU information (historical, by day)</td>
</tr>
<tr>
<td>CpuAggregateByHour</td>
<td>Aggregate CPU information (historical, by hour)</td>
</tr>
</tbody>
</table>
CpuAggregateByMinute | Aggregate CPU information (historical, by minute)
CpuAggregateBySecond | Aggregate CPU information (historical, by second)
CpuInfo | CPU information
CpuInfoByDay | CPU information (historical, by day)
CpuInfoByHour | CPU information (historical, by hour)
CpuInfoByMinute | CPU information (historical, by minute)
CpuInfoBySecond | CPU information (historical, by second)
DeploymentsCompleted | History of designs deployed
DesignsCompleted | History of designs executed
DiskResourceRejections | Disk Resource Rejection Records
Errors | History of all errors+warnings encountered
ExecutionEngineEvents | History of important events during local planning and execution
ExecutionEngineProfiles | History of EE profiles

(93 rows)

Related Topics

- Data Collector Functions
- Retaining Monitoring Information and How Vertica Calculates Database Size in the Administrator's Guide
- SET_DATA_COLLECTOR_TIME_POLICY
- SET_DATA_COLLECTOR_POLICY

DATABASE_BACKUPS

Lists historical information for each backup that successfully completed after running the vbr utility. This information is useful for determining whether to create a new backup before you advance the AHM. Because this system table displays historical information, its contents do not always reflect the current state of a backup repository. For example, if you delete a backup from a repository, the DATABASE_BACKUPS system table continues to display information about it.

To list existing backups, run vbr as described in Viewing Backups in the Administrator's Guide.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The timestamp of the backup.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the initiator node that performed the backup logging.</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
SNAPSHOT_NAME | VARCHAR | The name of the backup, as specified in the snapshotName parameter of the vbr configuration file.
BACKUP_EPOCH | INTEGER | The database epoch at which the backup was saved.
NODE_COUNT | INTEGER | The number of nodes backed up in the completed backup, and as listed in the [Mappingn] sections of the configuration file.
OBJECTS | VARCHAR | The name of the object(s) contained in an object-level backup. This column is empty if the record is for a full cluster backup.
FILE_SYSTEM_TYPE | VARCHAR | The type of file system, such as Linux.

### Privileges
Superuser

## DATABASE_CONNECTIONS

Lists the connections to other databases for importing and exporting data. See Importing and Exporting Data Across Databases in the Administrator's Guide.

### Column Name | Data Type | Description
--- | --- | ---
DATABASE | VARCHAR | The name of the connected database
USERNAME | VARCHAR | The username used to create the connection
HOST | VARCHAR | The host name used to create the connection
PORT | VARCHAR | The port number used to create the connection
ISVALID | BOOLEAN | Whether the connection is still open and usable or not
Example

```sql
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON '10.10.20.150',5433;
CONNECT
=> SELECT * FROM DATABASE_CONNECTIONS;
```

<table>
<thead>
<tr>
<th>database</th>
<th>username</th>
<th>host</th>
<th>port</th>
<th>isvalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmart</td>
<td>dbadmin</td>
<td>10.10.20.150</td>
<td>5433</td>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)

### DELETE_VECTOR

Holds information on deleted rows to speed up the delete process.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node storing the deleted rows.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema where the deleted rows are located.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The name of the projection where the deleted rows are located.</td>
</tr>
<tr>
<td>STORAGE_TYPE</td>
<td>VARCHAR</td>
<td>The type of storage containing the delete vector (WOS or ROS).</td>
</tr>
<tr>
<td>DV_OID</td>
<td>INTEGER</td>
<td>The unique numeric ID (OID) that identifies this delete vector.</td>
</tr>
<tr>
<td>STORAGE_OID</td>
<td>INTEGER</td>
<td>The unique numeric ID (OID) that identifies the storage container that holds the delete vector.</td>
</tr>
<tr>
<td>SAL_STORAGE_ID</td>
<td>VARCHAR</td>
<td>Unique hexadecimal numeric ID assigned by the Vertica catalog, which identifies the storage.</td>
</tr>
<tr>
<td>DELETED_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows deleted.</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes used to store the deletion.</td>
</tr>
<tr>
<td>START_EPOCH</td>
<td>INTEGER</td>
<td>The start epoch of the data in the delete vector.</td>
</tr>
<tr>
<td>END_EPOCH</td>
<td>INTEGER</td>
<td>The end epoch of the data in the delete vector.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IS_SORTED</td>
<td>BOOLEAN</td>
<td>Whether the storage container's data is sorted (WOS containers only).</td>
</tr>
</tbody>
</table>

**DEPLOY_STATUS**

Records the history of deployed Database Designer designs and their deployment steps.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIME</td>
<td>TIMESTAMP</td>
<td>Time when the row recorded the event.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user who deployed a design at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>DEPLOY_NAME</td>
<td>VARCHAR</td>
<td>Name the deployment, same as the user-specified design name.</td>
</tr>
<tr>
<td>DEPLOY_STEP</td>
<td>VARCHAR</td>
<td>Steps in the design deployment.</td>
</tr>
<tr>
<td>DEPLOY_STEP_STATUS</td>
<td>VARCHAR</td>
<td>Textual status description of the current step in the deploy process.</td>
</tr>
<tr>
<td>DEPLOY_STEP_COMPLETE_PERCENT</td>
<td>FLOAT</td>
<td>Progress of current step in percentage (0–100).</td>
</tr>
<tr>
<td>DEPLOY_COMPLETE_PERCENT</td>
<td>FLOAT</td>
<td>Progress of overall deployment in percentage (0–100).</td>
</tr>
<tr>
<td>ERROR_MESSAGE</td>
<td>VARCHAR</td>
<td>Error or warning message during deployment.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.
DEPLOYMENT_PROJECTION_STATEMENTS

Contains information about `CREATE PROJECTION` statements used to deploy a database design. Each row contains information about a different `CREATE PROJECTION` statement. The function `DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY` populates this table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPLOYMENT_ID</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the deployment.</td>
</tr>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Unique name that the user assigned to the design.</td>
</tr>
<tr>
<td>DEPLOYMENT_PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique ID assigned to the output projection by Database Designer.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique ID assigned to the statement type that creates the projection.</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>VARCHAR</td>
<td>Text for the statement that creates the projection.</td>
</tr>
</tbody>
</table>

DEPLOYMENT_PROJECTIONS

Contains information about projections created and dropped during the design. Each row contains information about a different projection. Database Designer populates this table after the design is deployed.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment_id</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the deployment.</td>
</tr>
<tr>
<td>deployment_projection_id</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the output projection.</td>
</tr>
<tr>
<td>design_name</td>
<td>VARCHAR</td>
<td>Name of the design being deployed.</td>
</tr>
<tr>
<td>deployment_projection_id</td>
<td>VARCHAR</td>
<td>Name that Database Designer assigned to the projection.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>anchor_table_schema</td>
<td>VARCHAR</td>
<td>Name of the schema that contains the table the projection is based on.</td>
</tr>
<tr>
<td>anchor_table_name</td>
<td>VARCHAR</td>
<td>Name of the table the projection is based on.</td>
</tr>
<tr>
<td>deployment_operation</td>
<td>VARCHAR</td>
<td>Action being taken on the projection, for example, add or drop.</td>
</tr>
</tbody>
</table>
| deployment_projection_type          | VARCHAR     | Indicates whether Database Designer has proposed new projections for this design (DBD) or is using the existing catalog design (CATALOG). The REENCODED suffix indicates that the projection sort order and segmentation are the same, but the projection columns have new encodings:  
  - DBD  
  - CATALOG  
  - DBD_REENCODED  
  - CATALOG_REENCODED |
| deploy_weight                       | INTEGER     | Weight of this projection in creating the design. This field is always 0 for projections that have been dropped. |
| estimated_size_on_disk              | INTEGER     | Approximate size of the projection on disk, in MB.                          |

**DEPOT_EVICTIONS**

Lists the details of the files that have been evicted from the depot when the database is in Eon Mode Beta.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVICTION_TIME</td>
<td>TIMESTAMP</td>
<td>The time a catalog subscription changed.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of a node where an eviction occurred.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the user.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The user who made changes to the depot.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session; these columns are useful for creating joins with other system tables.</td>
</tr>
<tr>
<td>REQUEST_ID</td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>STORAGE_ID</td>
<td>VARCHAR</td>
<td>The OID of the evicted file.</td>
</tr>
<tr>
<td>FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The size of the file in bytes that was evicted.</td>
</tr>
<tr>
<td>NUMBER_HITS</td>
<td>INTEGER</td>
<td>The number of times the file was accessed.</td>
</tr>
<tr>
<td>LAST_ACCESS_TIME</td>
<td>TIMESTAMP</td>
<td>The last time the file was read.</td>
</tr>
</tbody>
</table>

**DEPOT_FETCHES**

Lists the details about files the depot fetched when the database is in Eon Mode Beta.
### Table 1: DEPOT_UPLOADED Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Timestamp from the beginning of the fetch.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>Timestamp from the completion of the fetch.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that fetched from the depot.</td>
</tr>
<tr>
<td>PLAN_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>STORAGE_ID</td>
<td>VARCHAR</td>
<td>The OID of the fetched file.</td>
</tr>
<tr>
<td>FILE_SIZE(Bytes)</td>
<td>INTEGER</td>
<td>The fetch size in bytes.</td>
</tr>
<tr>
<td>SOURCE_FILE</td>
<td>VARCHAR</td>
<td>The source file path used.</td>
</tr>
<tr>
<td>DESTINATION_PATH</td>
<td>VARCHAR</td>
<td>The destination file path.</td>
</tr>
<tr>
<td>SOURCE_NODE</td>
<td>VARCHAR</td>
<td>The source node used.</td>
</tr>
</tbody>
</table>

### DEPOT_UPLOADED

Lists the details about uploads to communal storage from the depot.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Timestamp from the beginning of the upload.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>Timestamp from the completion of the upload.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of a node where an upload occurred.</td>
</tr>
<tr>
<td>PLAN_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>FILE_SIZE(Bytes)</td>
<td>INTEGER</td>
<td>The size of the file in bytes that was uploaded.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SOURCE_FILE</td>
<td>VARCHAR</td>
<td>The source file path used.</td>
</tr>
<tr>
<td>DESTINATION_FILE</td>
<td>VARCHAR</td>
<td>The destination file path.</td>
</tr>
</tbody>
</table>

**DESIGN_QUERIES**

Contains info about design queries for a given design. The following functions populate this table:

- **DESIGNER_ADD DESIGN QUERIES**
- **DESIGNER_ADD DESIGN QUERIES FROM RESULTS**
- **DESIGNER_ADD DESIGN_QUERY**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGN_ID</td>
<td>INTEGER</td>
<td>Unique id that Database Designer assigned to the design.</td>
</tr>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Name that you specified for the design.</td>
</tr>
<tr>
<td>DESIGN_QUERY_ID</td>
<td>INTEGER</td>
<td>Unique id that Database Designer assigned to the design query.</td>
</tr>
<tr>
<td>DESIGN_QUERY_ID_INDEX</td>
<td>INTEGER</td>
<td>Database Designer chunks the query text if it exceeds the maximum attribute size before storing it in this table. Database Designer stored all chunks stored under the same value of DESIGN_QUERY_ID. DESIGN_QUERY_ID_INDEX keeps track of the order of the chunks, starting with 0 and ending in n, the index of the final chunk.</td>
</tr>
<tr>
<td>QUERY_TEXT</td>
<td>VARCHAR</td>
<td>Text of the query chunk, or the entire query text if it does not exceed the maximum attribute size.</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>FLOAT</td>
<td>A value from 0 to 1 that indicates the importance of that query in creating the</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DESIGN_QUERY_SEARCH_PATH</td>
<td>VARCHAR</td>
<td>The search path with which the query is to be parsed.</td>
</tr>
<tr>
<td>DESIGN_QUERY_SIGNATURE</td>
<td>INTEGER</td>
<td>Categorizes queries that affect the design that Database Designer creates in the same way. Database Designer assigns a signature to each query, weights one query for each signature group, depending on how many queries there are with that signature, and Database Designer considers that query when creating the design.</td>
</tr>
</tbody>
</table>

design. Assign a higher weight to queries that you run frequently so that Database Designer prioritizes those queries in creating the design. Default: 1.

---

**Example**

Add queries to VMART_DESIGN and query the DESIGN_QUERIES table:

```sql
=> SELECT DESIGNER_ADD DESIGN QUERIES ('VMART_DESIGN', '/tmp/examples/vmart_queries.sql', 'true');
DESIGNER_ADD DESIGN QUERIES

Number of accepted queries = 9
Number of queries referencing non-design tables = 0
Number of unsupported queries = 0
Number of illegal queries = 0

=> \x
Expanded display is on.
=> SELECT * FROM V_MONITOR.DESIGN.QUERIES

<table>
<thead>
<tr>
<th>RECORD 1</th>
<th>-----------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>vmart_design</td>
</tr>
<tr>
<td>design_query_id</td>
<td>1</td>
</tr>
<tr>
<td>design_query_id_index</td>
<td>0</td>
</tr>
<tr>
<td>query_text</td>
<td>SELECT fat_content FROM (SELECT DISTINCT fat_content FROM product_dimension WHERE department_description IN ('Dairy') ) AS food ORDER BY fat_content LIMIT 5;</td>
</tr>
<tr>
<td>weight</td>
<td>1</td>
</tr>
<tr>
<td>design_query_search_path</td>
<td>v_dbd_vmart_design_vmart_design_ltt, &quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
<tr>
<td>design_query_signature</td>
<td>45035996273724651</td>
</tr>
</tbody>
</table>
```
### DESIGN_STATUS

Records the progress of a running Database Designer design or history of the last Database Designer design executed by the current user.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIME</td>
<td>TIMESTAMP</td>
<td>Time when the row recorded the event.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user who ran a design at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Name of the user-specified design.</td>
</tr>
<tr>
<td>DESIGN_PHASE</td>
<td>VARCHAR</td>
<td>Phase of the design.</td>
</tr>
<tr>
<td>PHASE_STEP</td>
<td>VARCHAR</td>
<td>Substep in each design phase.</td>
</tr>
<tr>
<td>PHASE_STEP_COMPLETE_PERCENT</td>
<td>FLOAT</td>
<td>Progress of current substep in percentage (0–100).</td>
</tr>
<tr>
<td>PHASE_COMPLETE_PERCENT</td>
<td>FLOAT</td>
<td>Progress of current design phase in percentage (0–100).</td>
</tr>
</tbody>
</table>
Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Example

The following example shows the content of the DESIGN_STATUS table of a complete Database Designer run:

```
=> SELECT event_time, design_name, design_phase, phase_complete_percent
    FROM v_monitor.design_status;

  event_time | design_name | design_phase                      | phase_complete_percent |
  -------------------+-------------+-----------------------------------+------------------------
   2012-02-14 10:31:20 | design1     | Design started                    |                        |
   2012-02-14 10:31:21 | design1     | Design in progress: Analyze statistics phase | 33.33                 |
   2012-02-14 10:31:22 | design1     | Analyzing data statistics         | 66.67                 |
   2012-02-14 10:31:24 | design1     | Analyzing data statistics         | 100                   |
   2012-02-14 10:31:25 | design1     | Design in progress: Query optimization phase |                        |
   2012-02-14 10:31:25 | design1     | Optimizing query performance      | 37.5                  |
   2012-02-14 10:31:31 | design1     | Optimizing query performance      | 62.5                  |
   2012-02-14 10:31:36 | design1     | Optimizing query performance      | 75                    |
   2012-02-14 10:31:39 | design1     | Optimizing query performance      | 87.5                  |
   2012-02-14 10:31:41 | design1     | Optimizing query performance      | 87.5                  |
   2012-02-14 10:31:42 | design1     | Design in progress: Storage optimization phase |                        |
   2012-02-14 10:31:44 | design1     | Optimizing storage footprint      | 4.17                  |
   2012-02-14 10:31:44 | design1     | Optimizing storage footprint      | 16.67                 |
   2012-02-14 10:32:04 | design1     | Optimizing storage footprint      | 29.17                 |
   2012-02-14 10:32:04 | design1     | Optimizing storage footprint      | 31.25                 |
   2012-02-14 10:32:05 | design1     | Optimizing storage footprint      | 33.33                 |
   2012-02-14 10:32:05 | design1     | Optimizing storage footprint      | 35.42                 |
   2012-02-14 10:32:05 | design1     | Optimizing storage footprint      | 37.5                  |
   2012-02-14 10:32:05 | design1     | Optimizing storage footprint      | 62.5                  |
   2012-02-14 10:32:05 | design1     | Optimizing storage footprint      | 87.5                  |
   2012-02-14 10:32:39 | design1     | Optimizing storage footprint      | 87.5                  |
   2012-02-14 10:32:41 | design1     | Optimizing storage footprint      | 100                   |
(24 rows)
```

**DESIGN_TABLES**

Contains information about all the design tables for all the designs for which you are the owner. Each row contains information about a different design table. Vertica creates this table when you run **DESIGNER_CREATE_DESIGN**.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Unique name that the user specified for the design.</td>
</tr>
<tr>
<td>DESIGN_TABLE_ID</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the design table.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the schema that contains the design table.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>System object identifier (OID) assigned to the design table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the design table.</td>
</tr>
</tbody>
</table>

**Example**

Add all the tables from the VMart database to the design VMART_DESIGN. This operation populates the DESIGN_TABLES table:

```sql
=> SELECT DESIGNER_ADD DESIGN_TABLES('VMART_DESIGN', 'online_sales.*');
DESIGNER_ADD DESIGN_TABLES
-------------------------------
3
(1 row)
=> SELECT DESIGNER_ADD DESIGN_TABLES('VMART_DESIGN', 'public.*');
DESIGNER_ADD DESIGN_TABLES
-------------------------------
9
(1 row)
=> SELECT DESIGNER_ADD DESIGN_TABLES('VMART_DESIGN', 'store.*');
DESIGNER_ADD DESIGN_TABLES
-------------------------------
3
(1 row)
=> SELECT * FROM DESIGN_TABLES;
design name    | design table id | table schema | table id | table name                                 |
----------------|-----------------|-------------|----------|-------------------------------------------|
VMART_DESIGN   | 1   | online_sales | 45035996373718754 | online_page_dimension    |
VMART_DESIGN   | 2   | online_sales | 45035996373718758 | call_center_dimension    |
VMART_DESIGN   | 3   | online_sales | 45035996373718762 | online_sales_fact        |
VMART_DESIGN   | 4   | public       | 45035996373718766 | customer_dimension       |
VMART_DESIGN   | 5   | public       | 45035996373718770 | product_dimension        |
VMART_DESIGN   | 6   | public       | 45035996373718774 | promotion_dimension      |
VMART_DESIGN   | 7   | public       | 45035996373718778 | date_dimension           |
VMART_DESIGN   | 8   | public       | 45035996373718782 | vendor_dimension         |
VMART_DESIGN   | 9   | public       | 45035996373718786 | employee_dimension       |
VMART_DESIGN   | 10  | public       | 45035996373718822 | shipping_dimension       |
VMART_DESIGN   | 11  | public       | 45035996373718826 | warehouse_dimension      |
VMART_DESIGN   | 12  | public       | 45035996373718830 | inventory_face           |
VMART_DESIGN   | 13  | store        | 45035996373718794 | store_dimension          |
```
**DESIGNS**

Contains information about a Database Designer design. After you create a design and specify certain parameters for Database Designer, `DESIGNER_CREATE_DESIGN` creates this table in the `V_MONITOR` schema.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGN_ID</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigns to this design.</td>
</tr>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Name that the user specifies for the design.</td>
</tr>
<tr>
<td>KSAFETY_LEVEL</td>
<td>INTEGER</td>
<td>K-safety level for the design. Database Designer assigns a K-safety value of 0 for clusters with 1 or 2 nodes, and assigns a value of 1 for clusters with 3 or more nodes.</td>
</tr>
<tr>
<td>OPTIMIZATION_OBJECTIVE</td>
<td>VARCHAR</td>
<td>Name of the optimization objective for the design. Valid values are:.Query, Load, Balanced (default).</td>
</tr>
<tr>
<td>DESIGN_TYPE</td>
<td>VARCHAR</td>
<td>Name of the design type. Valid values are: COMPREHENSIVE (default), INCREMENTAL.</td>
</tr>
<tr>
<td>PROPOSE_SUPER_FIRST</td>
<td>BOOLEAN</td>
<td>Specifies to propose superprojections before projections, by default false. If DESIGN_MODE is COMPREHENSIVE, this field has no impact.</td>
</tr>
<tr>
<td>DESIGN_AVAILABLE</td>
<td>BOOLEAN</td>
<td>t if the design is currently available, otherwise, f (default).</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COLLECTED_STATISTICS</td>
<td>BOOLEAN</td>
<td><code>t</code> if statistics are to be collected when creating the design, otherwise, <code>f</code> (default).</td>
</tr>
<tr>
<td>POPULATE_DESIGN_TABLES_FROM_QUERIES</td>
<td>BOOLEAN</td>
<td><code>t</code> if you want to populate the design tables from the design queries, otherwise, <code>f</code> (default).</td>
</tr>
<tr>
<td>ENCODING_DESIGN</td>
<td>BOOLEAN</td>
<td><code>t</code> if the design is an encoding optimization design on pre-existing projections, otherwise, <code>f</code> (default).</td>
</tr>
<tr>
<td>DEPLOYMENT_PARALLELISM</td>
<td>INTEGER</td>
<td>Number of tables to be deployed in parallel when the design is complete. Default: 0</td>
</tr>
<tr>
<td>PROPOSE_UNSEGMENTED_PROJECTIONS</td>
<td>BOOLEAN</td>
<td><code>t</code> if you specify unsegmented projections, otherwise, <code>f</code> (default).</td>
</tr>
<tr>
<td>ANALYZE_CORRELATIONS_MODE</td>
<td>INTEGER</td>
<td>Specifies how Database Designer should handle existing column correlations in a design and whether or not Database Designer should reanalyze existing column correlations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0—(default) Ignore column correlations when creating the design.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1—Consider the existing correlations in the tables when creating the design.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2—Analyze column correlations if not previously performed, and consider the column correlations when creating the design.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3—Analyze all column correlations in the tables and consider them when creating the design, even if they have been analyzed previously.</td>
</tr>
</tbody>
</table>
DISKRESOURCE_REJECTIONS

Returns requests for resources that are rejected due to disk space shortages. Output is aggregated by both RESOURCE_TYPE and REJECTED_REASON to provide more comprehensive information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>RESOURCE_TYPE</td>
<td>VARCHAR</td>
<td>The resource request requester (example: Temp files).</td>
</tr>
<tr>
<td>REJECTED_REASON</td>
<td>VARCHAR</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Insufficient disk space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Failed volume</td>
</tr>
<tr>
<td>REJECTED_COUNT</td>
<td>INTEGER</td>
<td>Number of times this REJECTED_REASON has been given for this RESOURCE_TYPE.</td>
</tr>
<tr>
<td>FIRST_REJECTED_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The time of the first rejection for this REJECTED_REASON and RESOURCE_TYPE.</td>
</tr>
<tr>
<td>LAST_REJECTED_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The time of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.</td>
</tr>
<tr>
<td>LAST_REJECTED_VALUE</td>
<td>INTEGER</td>
<td>The value of the most recent rejection for this REJECTED_REASON and RESOURCE_TYPE.</td>
</tr>
</tbody>
</table>

See Also

- RESOURCE_REJECTIONS
- CLEAR_RESOURCE_REJECTIONS
**DISK_STORAGE**

Returns the amount of disk storage used by the database on each node. Each node can have one or more storage locations, and the locations can be on different disks with separate properties, such as free space, used space, and block size. The information in this system table is useful in determining where data files reside.

All returned values for this system table are in the context of the file system of the host operating system, and are not specific to Vertica-specific space.

The storage usage annotation called CATALOG indicates that the location is used to store the catalog. Each CATALOG location is specified only when creating a new database. You cannot add a CATALOG location annotation using CREATE LOCATION, nor remove an existing CATALOG annotation.

**Storage Location Performance**

The performance of a storage location is measured with two values:

- Throughput in MB/sec
- Latency in seeks/sec

These two values are converted to a single number (Speed) with the following formula:

\[
\text{ReadTime (time to read 1MB)} = \frac{1}{\text{throughput}} + \frac{1}{\text{latency}}
\]

- \(1/\text{throughput}\) is the time taken to read 1MB of data
- \(1/\text{latency}\) is the time taken to seek to the data.
- ReadTime is the time taken to read 1MB of data.

A disk is faster than another disk if its ReadTime is less.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>STORAGE_PATH</td>
<td>VARCHAR</td>
<td>The path where the storage location is mounted.</td>
</tr>
<tr>
<td>STORAGE_USAGE</td>
<td>VARCHAR</td>
<td>The type of information stored in the location:</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DATA: Only data is stored in the location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TEMP: Only temporary files that are created during loads or queries are stored in the location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DATA,TEMP: Both types of files are stored in the location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• USER: The storage location can be used by non-dbadmin users, who are granted access to the storage location</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CATALOG: The area is used for the Vertica catalog. This usage is set internally and cannot be removed or changed.</td>
</tr>
<tr>
<td>RANK</td>
<td>INTEGER</td>
<td>The rank assigned to the storage location based on its performance. Ranks are used to create a storage locations on which projections, columns, and partitions are stored on different disks based on predicted or measured access patterns. See Managing Storage Locations in the Administrator's Guide.</td>
</tr>
<tr>
<td>THROUGHPUT</td>
<td>INTEGER</td>
<td>The measure of a storage location's performance in MB/sec. 1/throughput is the time taken to read 1MB of data.</td>
</tr>
<tr>
<td>LATENCY</td>
<td>INTEGER</td>
<td>The measure of a storage location's performance in seeks/sec. 1/latency is the time taken to seek to the data.</td>
</tr>
<tr>
<td>STORAGE_STATUS</td>
<td>VARCHAR</td>
<td>The status of the storage location: active or retired.</td>
</tr>
<tr>
<td>DISK_BLOCK_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The block size of the disk in bytes.</td>
</tr>
<tr>
<td>DISK_SPACE_USED_BLOCKS</td>
<td>INTEGER</td>
<td>The number of disk blocks in use.</td>
</tr>
<tr>
<td>DISK_SPACE_USED_MB</td>
<td>INTEGER</td>
<td>The number of megabytes of disk storage in use.</td>
</tr>
<tr>
<td>DISK_SPACE_FREE_BLOCKS</td>
<td>INTEGER</td>
<td>The number of free disk blocks available.</td>
</tr>
<tr>
<td>DISK_SPACE_FREE_MB</td>
<td>INTEGER</td>
<td>The number of megabytes of free storage available.</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
**DISK_SPACE_FREE_PERCENT** | VARCHAR | The percentage of free disk space remaining.

### ERROR_MESSAGES

Lists system error messages and warnings Vertica encounters while processing queries. Some errors occur when no transaction is in progress, so the transaction identifier or statement identifier columns might return NULL.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EVENT_TIMESTAMP</strong></td>
<td>TIMESTAMPTZ</td>
<td>Time when the row recorded the event.</td>
</tr>
<tr>
<td><strong>NODE_NAME</strong></td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td><strong>USER_ID</strong></td>
<td>INTEGER</td>
<td>Identifier of the user who received the error message.</td>
</tr>
<tr>
<td><strong>USER_NAME</strong></td>
<td>VARCHAR</td>
<td>Name of the user who received the error message at the time Vertica recorded the session.</td>
</tr>
<tr>
<td><strong>SESSION_ID</strong></td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td><strong>REQUEST_ID</strong></td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td><strong>TRANSACTION_ID</strong></td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td><strong>STATEMENT_ID</strong></td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td><strong>ERROR_LEVEL</strong></td>
<td>VARCHAR</td>
<td>Severity of the error, can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INFO</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
| NOTICE | | 
| WARNING | | 
| ERROR | | 
| ROLLBACK | | 
| INTERNAL | | 
| FATAL | | 
| PANIC | | 

**ERROR_CODE** INTEGER Error code that Vertica reports.

**MESSAGE** VARCHAR Textual output of the error message.

**DETAIL** VARCHAR Additional information about the error message, in greater detail.

**HINT** VARCHAR Actionable hint about the error. For example:

HINT: Set the locale in this session to en_US@collation=binary using the command "\locale en_US@collation=binary"

### Privileges
No explicit privileges are required. You only see the records for tables that you have privileges to view.

### EVENT_CONFIGURATIONS
Monitors the configuration of events.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_ID</td>
<td>VARCHAR</td>
<td>The name of the event.</td>
</tr>
<tr>
<td>EVENT_DELIVERY_</td>
<td>VARCHAR</td>
<td>The delivery channel on which the event</td>
</tr>
</tbody>
</table>
EXECUTION_ENGINE_PROFILES

Provides profiling information about query execution runs. The hierarchy of IDs, from highest level to actual execution is:

- PATH_ID
- BASEPLAN_ID
- LOCALPLAN_ID
- OPERATOR_ID

Counters (output from the COUNTER_NAME column) are collected for each actual Execution Engine (EE) operator instance.

The following columns combine to form a unique key for rows in EXECUTION_ENGINE_PROFILES:

- TRANSACTION_ID
- STATEMENT_ID
- NODE_NAME
- OPERATOR_ID
- COUNTER_NAME
- COUNTER_TAG

For more about profiling and debugging, see Profiling Database Performance in the Administrator's Guide.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node name for which information is listed.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the user.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>User name for which query profile information is listed.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier of the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed.</td>
</tr>
<tr>
<td>OPERATOR_NAME</td>
<td>VARCHAR</td>
<td>Name of the Execution Engine (EE) component; for example, NetworkSend.</td>
</tr>
<tr>
<td>OPERATOR_ID</td>
<td>INTEGER</td>
<td>Identifier assigned by the EE operator instance that performs the work. OPERATOR_ID is different from LOCALPLAN_ID because each logical operator, such as Scan, may be executed by multiple threads concurrently. Each thread operates on a different operator instance, which has its own ID.</td>
</tr>
<tr>
<td>BASEPLAN_ID</td>
<td>INTEGER</td>
<td>Assigned by the optimizer on the initiator to EE operators in the original base (EXPLAIN) plan. Each EE operator in the base plan gets a unique ID.</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>INTEGER</td>
<td>Identifier that Vertica assigns to a query operation or path; for example to a logical grouping operation that might be performed by multiple execution engine operators.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For each path, the same PATH ID is shared between the query plan (using EXPLAIN output) and in error messages that refer to joins.</td>
</tr>
<tr>
<td>LOCALPLAN_ID</td>
<td>INTEGER</td>
<td>Identifier assigned by each local executor while preparing for plan execution (local planning). Some operators in the base plan, such as the Root operator, which is connected to the client, do not run on all nodes. Similarly, certain operators, such as ExprEval, are</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>added and removed during local planning due to implementation details.</td>
</tr>
<tr>
<td>ACTIVITY_ID</td>
<td>INTEGER</td>
<td>Identifier of the plan activity.</td>
</tr>
<tr>
<td>RESOURCE_ID</td>
<td>INTEGER</td>
<td>Identifier of the plan resource.</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>VARCHAR</td>
<td>Name of the counter. See the &quot;COUNTER_NAME Values&quot; section below this table.</td>
</tr>
<tr>
<td>COUNTER_TAG</td>
<td>VARCHAR</td>
<td>String that uniquely identifies the counter for operators that might need</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to distinguish between different instances. For example, COUNTER_TAG is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>used to identify to which of the node bytes are being sent to or received</td>
</tr>
<tr>
<td></td>
<td></td>
<td>from for the NetworkSend operator.</td>
</tr>
<tr>
<td>COUNTER_VALUE</td>
<td>INTEGER</td>
<td>Value of the counter.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Indicates whether the profile is active or completed, where t is active and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f is completed.</td>
</tr>
</tbody>
</table>

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

COUNTER_NAME Values

The value of COUNTER_NAME can be any of the following:

<table>
<thead>
<tr>
<th>COUNTER_NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active threads</td>
<td>A counter of the LoadUnion operator, which indicates the number of input</td>
</tr>
<tr>
<td>blocks analyzed by SIPS</td>
<td>The number of data blocks analyzed by SIPS expression from the Scan operator.</td>
</tr>
<tr>
<td>expression</td>
<td></td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>blocks filtered by SIPS expression</td>
<td>The number of data blocks filtered by SIPS expression from the Scan operator.</td>
</tr>
<tr>
<td>blocks filtered by SIPS value lists</td>
<td>The number of data blocks filtered by SIPS sorted value lists from the Scan operator.</td>
</tr>
<tr>
<td>buffers spilled</td>
<td>[NetworkSend] Buffers spilled to disk by NetworkSend.</td>
</tr>
<tr>
<td>bytes read from cache</td>
<td>[DataSource] The number of bytes read from Vertica cache when an EE DataSource operator is reading from ROS containers.</td>
</tr>
<tr>
<td>bytes read from disk</td>
<td>[DataSource] The number of bytes read from disk when an EE DataSource operator is reading from ROS containers.</td>
</tr>
<tr>
<td>bytes received</td>
<td>[NetworkRecv] The number of bytes received over the network for query execution.</td>
</tr>
<tr>
<td>bytes sent</td>
<td>[NetworkSend] Size of data after encoding and compression sent over the network (actual network bytes).</td>
</tr>
<tr>
<td>bytes spilled</td>
<td>[NetworkSend] Bytes spilled to disk by NetworkSend.</td>
</tr>
<tr>
<td>bytes total</td>
<td>Only relevant to SendFiles operator (that is, recover-by-container plan) total number of bytes to send / receive.</td>
</tr>
<tr>
<td>clock time (us)</td>
<td>Real-time clock time spent processing the query, in microseconds.</td>
</tr>
<tr>
<td>completed merge phases</td>
<td>Number of merge phases already completed by an LSort or DataTarget operator. Compare to the total merge phases. Variants on this value include join inner completed merge phases.</td>
</tr>
<tr>
<td>cumulative size of raw temp data (bytes)</td>
<td>Total amount of temporary data the operator has written to files. Compare to cumulative size of temp files (bytes) to understand impact of</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>encoding and compression in an externalizing operator. Variants on this value include join inner cumulative size of raw temp files (bytes).</td>
<td></td>
</tr>
<tr>
<td>cumulative size of temp files (bytes)</td>
<td>For externalizing operators only, the total number of encoded and compressed temp data the operator has written to files. A sort operator might go through multiple merge phases, where at each pass sorted chunks of data are merged into fewer chunks. This counter remembers the cumulative size of all temp files past and present. Variants on this value include join inner cumulative size of temp files (bytes).</td>
</tr>
<tr>
<td>current allocated rid memory (bytes)</td>
<td>Per-rid memory tracking: current allocation amount under this rid.</td>
</tr>
<tr>
<td>current file handles</td>
<td>Number of files open.</td>
</tr>
<tr>
<td>current memory allocations (count)</td>
<td>Number of actual allocator calls made.</td>
</tr>
<tr>
<td>current memory capacity (bytes)</td>
<td>Amount of system memory held, which includes chunks which have been only partially consumed.</td>
</tr>
<tr>
<td>current memory overhead (bytes)</td>
<td>Memory consumed, for example, by debug headers. (Normally no overhead.)</td>
</tr>
<tr>
<td>current memory padding (bytes)</td>
<td>Memory padding for free list tiers (2^n bytes).</td>
</tr>
<tr>
<td>current memory requested (bytes)</td>
<td>Memory actually requested by the caller.</td>
</tr>
<tr>
<td>current size of temp files (bytes)</td>
<td>For externalizing operators only, the current size of the encoded and compressed temp data that the operator has written to files. Variants on this value include join inner current size of temp files (bytes).</td>
</tr>
<tr>
<td>current threads</td>
<td>Unused.</td>
</tr>
<tr>
<td>current unbalanced memory</td>
<td>Pooled version of &quot;current memory XXX&quot; counters.</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>allocations (count)</td>
<td></td>
</tr>
<tr>
<td>current unbalanced memory capacity (bytes)</td>
<td></td>
</tr>
<tr>
<td>current unbalanced memory overhead (bytes)</td>
<td></td>
</tr>
<tr>
<td>current unbalanced memory requested (bytes)</td>
<td></td>
</tr>
<tr>
<td>distinct value estimation time (µs)</td>
<td>[Analyze Statistics] Time spent estimating the number of distinct values from the sample after data has been read off disk and into the statistical sample.</td>
</tr>
<tr>
<td>encoded bytes received</td>
<td>[NetworkRecv] Size of received data after decompressed (but still encoded) received over the network.</td>
</tr>
<tr>
<td>encoded bytes sent</td>
<td>[NetworkSend] Size of data sent over the network after encoding.</td>
</tr>
<tr>
<td>end time</td>
<td>Time (timestamp) when Vertica stopped processing the operation</td>
</tr>
<tr>
<td>estimated rows produced</td>
<td>Number of rows that the optimizer estimated would be produced. See rows produced for the actual number of rows that are produced.</td>
</tr>
<tr>
<td>exceptions cumulative size of raw temp data (bytes)</td>
<td>Counters that store the total or current size of exception data.</td>
</tr>
<tr>
<td>exceptions rows cumulative size of temp files (bytes)</td>
<td></td>
</tr>
<tr>
<td>exceptions rows current size of temp files (bytes)</td>
<td></td>
</tr>
<tr>
<td>execution time (us)</td>
<td>CPU clock time spent processing the query, in microseconds.</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fast aggregated rows</td>
<td>The number of rows being processed by fast aggregations in the hash groupby operator (no group/aggregation).</td>
</tr>
<tr>
<td>file handles</td>
<td>The number of file handles in use for an operator. Deprecated. See peak file handles or current file handles.</td>
</tr>
<tr>
<td>files completed</td>
<td>Relevant only to SendFiles/RecvFiles operators (that is, recover-by-container plan) number of files sent / received.</td>
</tr>
<tr>
<td>files total</td>
<td>Relevant only to SendFiles/RecvFiles operators (that is, recover-by-container plan) total number of files to send / receive.</td>
</tr>
<tr>
<td>Hadoop FS bytes read through native libhdfs++ client</td>
<td>[Scan, Load] The number of bytes read from an hdfs source (using libhdfs++).</td>
</tr>
<tr>
<td>Hadoop FS bytes read through webhdfs</td>
<td>[Scan, Load] The number of bytes read from a webhdfs source.</td>
</tr>
<tr>
<td>Hadoop FS bytes written through webhdfs</td>
<td>[DataTarget] The number of bytes written to webhdfs storage.</td>
</tr>
<tr>
<td>Hadoop FS hdfs:// operations that used native libhdfs++ calls</td>
<td>[Scan, Load, DataTarget] The number of times Vertica opened a file with an hdfs:// URL and used the native hdfs protocol</td>
</tr>
<tr>
<td>Hadoop FS hdfs:// operations that used webhdfs calls</td>
<td>[Scan, Load, DataTarget] The number of times Vertica opened a file with an hdfs:// URL and used the webhdfs protocol</td>
</tr>
<tr>
<td>Hadoop FS read operations through native libhdfs++ client failure count</td>
<td>[Scan, Load] The number of times a native libhdfs++ source encountered an error and gave up</td>
</tr>
<tr>
<td>Hadoop FS read operations through native libhdfs++ client retry count</td>
<td>[Scan, Load] The number of times a native libhdfs++ source encountered an error and retried</td>
</tr>
<tr>
<td>Hadoop FS read operations</td>
<td>[Scan, Load] The number of times a webhdfs source...</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>through webhdfs failure count</td>
<td>encountered an error and gave up</td>
</tr>
<tr>
<td>Hadoop FS read operations through webhdfs retry count</td>
<td>[Scan, Load] The number of times a webhdfs source encountered an error and retried</td>
</tr>
<tr>
<td>Hadoop FS write operations through webhdfs failure count</td>
<td>[DataTarget] The number of times a webhdfs write encountered an error and gave up</td>
</tr>
<tr>
<td>Hadoop FS write operations through webhdfs retry count</td>
<td>[DataTarget] The number of times a webhdfs write encountered an error and retried</td>
</tr>
<tr>
<td>histogram creation time (us)</td>
<td>[Analyze Statistics] Time spent estimating the number of distinct values from the sample after data has been read off disk and into the statistical sample.</td>
</tr>
<tr>
<td>initialization time (us)</td>
<td>The time in microseconds spent initializing an operator during the CompilePlan step of query processing. For example, initialization time could include the time spent compiling expressions and gathering resources.</td>
</tr>
<tr>
<td>input queue wait (μs)</td>
<td>Time in microseconds that an operator spends waiting for upstream operators.</td>
</tr>
<tr>
<td>input rows</td>
<td>Actual number of rows that were read into the operator.</td>
</tr>
<tr>
<td>input size (bytes)</td>
<td>Total number of bytes of the Load operator's input source, where NULL is unknown (read from FIFO).</td>
</tr>
<tr>
<td>inputs processed</td>
<td>The number of sources processed by a Load operator.</td>
</tr>
<tr>
<td>intermediate rows to process</td>
<td>The number of rows to be processed in a phase as determined by a sort or GROUP BY (HASH).</td>
</tr>
<tr>
<td>join inner clock time (us)</td>
<td>The real clock time spending on processing the inner input of the join operator.</td>
</tr>
<tr>
<td>join inner completed mergephases</td>
<td>See the completed merge phases counter.</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>join inner cumulative size of raw temp data (bytes)</td>
<td>The CPU clock time spent on processing the inner input of the join operator.</td>
</tr>
<tr>
<td>join inner cumulative size of temp files (bytes)</td>
<td>The time spent for building the hash table for the inner input of the join operator.</td>
</tr>
<tr>
<td>join inner current size of temp files (bytes)</td>
<td>The number of hash table collisions that occurred when building the hash table for the inner input of the join operator.</td>
</tr>
<tr>
<td>join inner execution time (us)</td>
<td>The number of hash table entries for the inner input of the join operator.</td>
</tr>
<tr>
<td>join inner hash table building time (us)</td>
<td>See the completed merge phases counter.</td>
</tr>
<tr>
<td>join inner hash table collisions</td>
<td>The number of cancel requests received (per operator)</td>
</tr>
<tr>
<td>join outer clock time (us)</td>
<td>The CPU clock time spent on processing the outer input of the join operator (including doing the join).</td>
</tr>
<tr>
<td>join outer execution time (us)</td>
<td>The real clock time spent on processing the outer input of the join operator (including doing the join).</td>
</tr>
<tr>
<td>max sample size (rows)</td>
<td>[Analyze Statistics] Maximum number of rows that will be stored in the statistical sample.</td>
</tr>
<tr>
<td>memory allocated (bytes)</td>
<td>Memory allocated by this operator. Deprecated.</td>
</tr>
<tr>
<td>memory reserved (bytes)</td>
<td>Memory reserved by this operator. Deprecated.</td>
</tr>
<tr>
<td>network wait (us)</td>
<td>[NetworkSend, NetworkRecv] Time in microseconds spent waiting on the network.</td>
</tr>
<tr>
<td>number of cancel requests</td>
<td>The number of cancel requests received (per operator)</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>received</td>
<td>when cancelling a call to the execution engine.</td>
</tr>
<tr>
<td>number of invocations</td>
<td>The number of times a UDSF function was invoked.</td>
</tr>
<tr>
<td>output queue wait (us)</td>
<td>Time in microseconds that an operator spends waiting for the output buffer to be consumed by a downstream operator.</td>
</tr>
<tr>
<td>peak allocated rid memory (bytes)</td>
<td>Per-rid memory tracking: peak allocation amount under this rid.</td>
</tr>
<tr>
<td>peak cooperating threads</td>
<td>Peak number of threads which parsed (in parallel) a single load source, using &quot;cooperative parse.&quot; counter_tag indicates the source when joining with dc_load_events.</td>
</tr>
<tr>
<td>peak file handles</td>
<td>Peak value of the corresponding &quot;current XXX&quot; counters.</td>
</tr>
<tr>
<td>peak memory allocations (count)</td>
<td></td>
</tr>
<tr>
<td>peak memory capacity (bytes)</td>
<td></td>
</tr>
<tr>
<td>peak memory overhead (bytes)</td>
<td></td>
</tr>
<tr>
<td>peak memory padding (bytes)</td>
<td></td>
</tr>
<tr>
<td>peak memory requested (bytes)</td>
<td></td>
</tr>
<tr>
<td>peak temp space</td>
<td></td>
</tr>
<tr>
<td>peak threads</td>
<td></td>
</tr>
<tr>
<td>peak unbalanced memory allocations (count)</td>
<td></td>
</tr>
<tr>
<td>peak unbalanced memory capacity (bytes)</td>
<td></td>
</tr>
<tr>
<td>peak unbalanced memory overhead (bytes)</td>
<td></td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>peak unbalanced memory padding (bytes)</td>
<td>Offset value of a portion descriptor in an apportioned load. counter_tag indicates the source when joining with dc_load_events.</td>
</tr>
<tr>
<td>peak unbalanced memory requested (bytes)</td>
<td>Size value of a portion descriptor in an apportioned load. counter_tag indicates the source when joining with dc_load_events.</td>
</tr>
<tr>
<td>portion offset</td>
<td>[NetworkSend] Time in microseconds spent by NetworkSend when stalled waiting for network buffers to clear.</td>
</tr>
<tr>
<td>portion size</td>
<td>[NetworkSend] Time in microseconds spent by the input operator making rows to send.</td>
</tr>
<tr>
<td>read (bytes)</td>
<td>Number of bytes read from the input source by the Load operator.</td>
</tr>
<tr>
<td>receive time (us)</td>
<td>Time in microseconds that aRecv operator spends reading data from its socket.</td>
</tr>
<tr>
<td>rejected data cumulative size of raw temp data (bytes)</td>
<td>Counters that store total or current size of rejected row numbers. Are variants of:</td>
</tr>
<tr>
<td>rejected data cumulative size of temp files (bytes)</td>
<td>• cumulative size of raw temp data (bytes)</td>
</tr>
<tr>
<td>rejected data current size of temp files (bytes)</td>
<td>• cumulative size of temp files (bytes)</td>
</tr>
<tr>
<td>rejected rows cumulative size of raw temp data (bytes)</td>
<td>• current size of temp files (bytes)</td>
</tr>
<tr>
<td>rejected rows cumulative</td>
<td></td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>size of temp files (bytes)</td>
<td></td>
</tr>
<tr>
<td>rejected rows current size of temp files (bytes)</td>
<td></td>
</tr>
<tr>
<td>reserved rid memory (bytes)</td>
<td>Per-rid memory tracking: total memory reservation under this rid.</td>
</tr>
<tr>
<td>rle rows produced</td>
<td>Number of physical tuples produced by an operator.</td>
</tr>
<tr>
<td></td>
<td>Complements the rows produced counter, which shows the number of logical</td>
</tr>
<tr>
<td></td>
<td>rows produced by an operator.</td>
</tr>
<tr>
<td></td>
<td>For example, if a value occurs 1000 rows consecutively and is RLE</td>
</tr>
<tr>
<td></td>
<td>encoded, it counts as 1000 rows produced not only 1 rle rows produced.</td>
</tr>
<tr>
<td>ROS blocks bounded</td>
<td>[DataTarget] Number of ROS blocks created, due to boundary alignment with</td>
</tr>
<tr>
<td></td>
<td>RLE prefix columns, when an EE DataTarget operator is writing to ROS</td>
</tr>
<tr>
<td></td>
<td>containers.</td>
</tr>
<tr>
<td>ROS blocks encoded</td>
<td>[DataTarget] Number of ros blocks created when an EE DataTarget operator is</td>
</tr>
<tr>
<td></td>
<td>writing to ROS containers.</td>
</tr>
<tr>
<td>ROS bytes written</td>
<td>[DataTarget] Number of bytes written to disk when an EE DataTarget operator</td>
</tr>
<tr>
<td></td>
<td>is writing to ROS containers.</td>
</tr>
<tr>
<td>rows filtered by SIPS expression</td>
<td>The number of rows filtered by the SIPS expression from the Scan operator.</td>
</tr>
<tr>
<td>rows in sample</td>
<td>[Analyze Statistics] Actual number of rows that will be stored in the</td>
</tr>
<tr>
<td></td>
<td>statistical sample.</td>
</tr>
<tr>
<td>rows output by sort</td>
<td>[DataTarget] Number of rows sorted when an EE DataTarget operator is writing</td>
</tr>
<tr>
<td></td>
<td>to ROS containers.</td>
</tr>
<tr>
<td>rows processed</td>
<td>[DataSource] Number of rows processed when an EE DataSource operator is</td>
</tr>
<tr>
<td></td>
<td>reading from ROS containers.</td>
</tr>
<tr>
<td>rows processed by SIPS expression</td>
<td>The number of rows processed by the SIPS expression in the Scan operator.</td>
</tr>
<tr>
<td>rows produced</td>
<td>Number of logical rows produced by an operator. See</td>
</tr>
<tr>
<td>COUNTER_NAME</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>also the rle rows produced counter.</td>
<td></td>
</tr>
<tr>
<td>rows pruned by valindex</td>
<td>[DataSource] Number of rows it skips direct scanning with help of valindex when an EE DataSource operator is writing to ROS containers. This counter's value is not greater than &quot;rows processed&quot; counter.</td>
</tr>
<tr>
<td>rows read in sort</td>
<td>See the counter, total rows read in sort.</td>
</tr>
<tr>
<td>rows received</td>
<td>[NetworkRecv] Number of received sent over the network.</td>
</tr>
<tr>
<td>rows rejected</td>
<td>The number of rows rejected by the Load operator.</td>
</tr>
<tr>
<td>rows sent</td>
<td>[NetworkSend] Number of rows sent over the network.</td>
</tr>
<tr>
<td>rows to process</td>
<td>The total number of rows to be processed in a phase, based upon the number of table accesses. Compare to the counter, rows processed. Divide the rows processed value by the rows to process value for percent completion.</td>
</tr>
<tr>
<td>rows written in join sort</td>
<td>The total number of rows being read out of the sort facility in Join.</td>
</tr>
<tr>
<td>rows written in sort</td>
<td>The number of rows read out of the sort by the SortManager. This counter and the counter total rows read from sort are typically equal.</td>
</tr>
<tr>
<td>send time (us)</td>
<td>Time in microseconds that a Send operator spends writing data to its socket.</td>
</tr>
<tr>
<td>start time</td>
<td>Time (timestamp) when Vertica started to process the operation.</td>
</tr>
<tr>
<td>total merge phases</td>
<td>Number of merge phases an LSort or DataTarget operator must complete to finish sorting its data. NULL until the operator can compute this value (all data must first be ingested by the operator). Variants on this value include join inner total merge phases.</td>
</tr>
<tr>
<td>total rows read in join</td>
<td>The total number of rows being put into the sort facility</td>
</tr>
</tbody>
</table>
### COUNTER_NAME | Description
--- | ---
sort | in Join.
total rows read in sort | The total number of rows ingested into the sort by the SortManager. This counter and the counter rows written in sort are typically equal.
total rows written in sort | See the counter, rows written in sort.
total sources | Total number of distinct input sources processed in a load.
unpacked (bytes) | The number of bytes produced by a compressed source in a load (for example, for a gzip file, the size of the file when decompressed).
wait clock time (us) | StorageUnion wait time in microseconds.
WOS bytes acquired | Number of bytes acquired from the WOS by a DataTarget operator.  
  **Note:** This is usually more but can be less than WOS bytes written if an earlier statement in the transaction acquired some WOS memory.
WOS bytes written | Number of bytes written to the WOS by a DataTarget operator.
written rows | [DataTarget] Number of rows written when an EE DataTarget operator writes to ROS containers

### Examples

The two queries below show the contents of the `EXECUTION_ENGINE_PROFILES` table:

```sql
=> SELECT operator_name, operator_id, counter_name, counter_value  
    FROM EXECUTION_ENGINE_PROFILES WHERE operator_name = 'Scan'  
    ORDER BY counter_value DESC;
```

<table>
<thead>
<tr>
<th>operator_name</th>
<th>operator_id</th>
<th>counter_name</th>
<th>counter_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>12</td>
<td>end time</td>
<td>397916465478595</td>
</tr>
<tr>
<td>Scan</td>
<td>9</td>
<td>end time</td>
<td>397916465478510</td>
</tr>
<tr>
<td>Scan</td>
<td>12</td>
<td>start time</td>
<td>397916465462098</td>
</tr>
</tbody>
</table>
```sql
SELECT DISTINCT counter_name FROM execution_engine_profiles;
```

<table>
<thead>
<tr>
<th>operator_name</th>
<th>path_id</th>
<th>counter_name</th>
<th>counter_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join</td>
<td>1</td>
<td>estimated rows produced</td>
<td>10000</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>file handles</td>
<td>0</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>memory allocated (bytes)</td>
<td>2405824</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>memory reserved (bytes)</td>
<td>1769472</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>rle rows produced</td>
<td>3</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>rows produced</td>
<td>3</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>clock time (us)</td>
<td>24105</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>execution time (us)</td>
<td>235</td>
</tr>
</tbody>
</table>

(24 rows)

The following query includes the `path_id` column, which links the path that the query optimizer takes (via the EXPLAIN command’s textual output) with join error messages.

```sql
SELECT operator_name, path_id, counter_name, counter_value FROM execution_engine_profiles;
```
EXTERNAL_TABLE_DETAILS

Returns the amount of disk storage used by the source files backing external tables in the database. The information in this system table is useful in determining Hadoop license compliance.

When computing the size of an external table, Vertica counts all data found in the location specified by the COPY FROM clause. If you have a directory that contains ORC and delimited files, for example, and you define your external table with "COPY FROM *" instead of "COPY FROM *.orc", this table includes the size of the delimited files. (You would probably also encounter errors when querying that external table.) When you query this system table Vertica does not validate your table definition; it just uses the path to find files to report.

Restrict your queries to filter by schema, table, or format to avoid expensive queries. Vertica calculates the values in this table at query time, so "SELECT *" accesses every input file contributing to every external table.

Predicates in queries may use only the TABLE_SCHEMA, TABLE_NAME, and SOURCE_FORMAT columns. Values are case-sensitive.

This table includes TEMP external tables.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_OID</td>
<td>INTEGER</td>
<td>The unique identification number of the schema in which the external table resides.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema in which the external table resides.</td>
</tr>
<tr>
<td>TABLE_OID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>INTEGER</td>
<td>The table name.</td>
</tr>
<tr>
<td>SOURCE_FORMAT</td>
<td>VARCHAR</td>
<td>The data format the source file used, one of ORC, PARQUET, DELIMITED, USER DEFINED, or NULL if another format.</td>
</tr>
<tr>
<td>TOTAL_FILE_COUNT</td>
<td>INTEGER</td>
<td>The number of files used to store this table's data, expanding globs and partitions.</td>
</tr>
<tr>
<td>TOTAL_FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>Total number of bytes used by all of this table's data</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>files.</td>
</tr>
<tr>
<td>SOURCE_STATEMENT</td>
<td>VARCHAR</td>
<td>The load statement used to copy data from the source files.</td>
</tr>
<tr>
<td>FILE_ACCESS_ERROR</td>
<td>VARCHAR</td>
<td>The access error returned during the source statement. NULL, if there was no access error during the source statement.</td>
</tr>
</tbody>
</table>

**HOST_RESOURCES**

Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_NAME</td>
<td>VARCHAR</td>
<td>The host name for which information is listed.</td>
</tr>
<tr>
<td>OPEN_FILES_LIMIT</td>
<td>INTEGER</td>
<td>The maximum number of files that can be open at one time on the node.</td>
</tr>
<tr>
<td>THREADS_LIMIT</td>
<td>INTEGER</td>
<td>The maximum number of threads that can coexist on the node.</td>
</tr>
<tr>
<td>CORE_FILE_LIMIT_MAX_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The maximum core file size allowed on the node.</td>
</tr>
<tr>
<td>PROCESSOR_COUNT</td>
<td>INTEGER</td>
<td>The number of system processors.</td>
</tr>
<tr>
<td>PROCESSOR_CORE_COUNT</td>
<td>INTEGER</td>
<td>The number of processor cores in the system.</td>
</tr>
<tr>
<td>PROCESSOR_DESCRIPTION</td>
<td>VARCHAR</td>
<td>A description of the processor. For example: Inter(R) Core(TM)2 Duo CPU T8100 @2.10GHz (1 row)</td>
</tr>
<tr>
<td>OPENED_FILE_COUNT</td>
<td>INTEGER</td>
<td>The total number of open files on the node.</td>
</tr>
<tr>
<td>OPENED_SOCKET_COUNT</td>
<td>INTEGER</td>
<td>The total number of open sockets on the node.</td>
</tr>
<tr>
<td>OPENED_NONFILE_NONSOCKET_COUNT</td>
<td>INTEGER</td>
<td>The total number of other file descriptions open in which 'other' could be a directory or</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>host_name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open_files_limit</td>
<td>INTEGER</td>
<td>The total amount of physical RAM, in bytes, available on the system.</td>
</tr>
<tr>
<td>threads_limit</td>
<td>INTEGER</td>
<td>The amount of physical RAM, in bytes, left unused by the system.</td>
</tr>
<tr>
<td>core_file_limit_max_size_bytes</td>
<td>INTEGER</td>
<td>The amount of physical RAM, in bytes, used for file buffers on the system.</td>
</tr>
<tr>
<td>processor_count</td>
<td>INTEGER</td>
<td>The amount of physical RAM, in bytes, used as cache memory on the system.</td>
</tr>
<tr>
<td>total_swap_memory_bytes</td>
<td>INTEGER</td>
<td>The total amount of swap memory available, in bytes, on the system.</td>
</tr>
<tr>
<td>total_swap_memory_free_bytes</td>
<td>INTEGER</td>
<td>The total amount of swap memory free, in bytes, on the system.</td>
</tr>
<tr>
<td>disk_space_free_mb</td>
<td>INTEGER</td>
<td>The free disk space available, in megabytes, for all storage location file systems (data directories).</td>
</tr>
<tr>
<td>disk_space_used_mb</td>
<td>INTEGER</td>
<td>The disk space used, in megabytes, for all storage location file systems.</td>
</tr>
<tr>
<td>disk_space_total_mb</td>
<td>INTEGER</td>
<td>The total free disk space available, in megabytes, for all storage location file systems.</td>
</tr>
</tbody>
</table>

**Example**

```
=> SELECT * FROM HOST_RESOURCES;
- [ RECORD 1 ]---------------------------------------------------------------
  host_name | 10.20.100.247
  open_files_limit | 65536
  threads_limit     | 3833
  core_file_limit_max_size_bytes | 0
  processor_count   | 2
  processor_core_count | 2
  processor_description | Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
  opened_file_count  | 8
  opened_socket_count | 15
```
<table>
<thead>
<tr>
<th>opened_nonfile_nonsocket_count</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_memory_bytes</td>
<td>4018823168</td>
</tr>
<tr>
<td>total_memory_free_bytes</td>
<td>126550016</td>
</tr>
<tr>
<td>total_buffer_memory_bytes</td>
<td>191083392</td>
</tr>
<tr>
<td>total_memory_cache_bytes</td>
<td>2418753536</td>
</tr>
<tr>
<td>total_swap_memory_bytes</td>
<td>2147479552</td>
</tr>
<tr>
<td>total_swap_memory_free_bytes</td>
<td>2145771520</td>
</tr>
<tr>
<td>disk_space_free_mb</td>
<td>18238</td>
</tr>
<tr>
<td>disk_space_used_mb</td>
<td>30013</td>
</tr>
<tr>
<td>disk_space_total_mb</td>
<td>48251</td>
</tr>
</tbody>
</table>

- [ RECORD 2 ]----------------------------------------

<table>
<thead>
<tr>
<th>host_name</th>
<th>10.20.100.248</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_files_limit</td>
<td>65536</td>
</tr>
<tr>
<td>threads_limit</td>
<td>3833</td>
</tr>
<tr>
<td>core_file_limit_max_size_bytes</td>
<td>0</td>
</tr>
<tr>
<td>processor_count</td>
<td>2</td>
</tr>
<tr>
<td>processor_core_count</td>
<td>2</td>
</tr>
<tr>
<td>processor_description</td>
<td>Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz</td>
</tr>
<tr>
<td>opened_file_count</td>
<td>8</td>
</tr>
<tr>
<td>opened_socket_count</td>
<td>9</td>
</tr>
<tr>
<td>opened_nonfile_nonsocket_count</td>
<td>7</td>
</tr>
<tr>
<td>total_memory_bytes</td>
<td>4018823168</td>
</tr>
<tr>
<td>total_memory_free_bytes</td>
<td>356466688</td>
</tr>
<tr>
<td>total_buffer_memory_bytes</td>
<td>327278592</td>
</tr>
<tr>
<td>total_memory_cache_bytes</td>
<td>2744279040</td>
</tr>
<tr>
<td>total_swap_memory_bytes</td>
<td>2147479552</td>
</tr>
<tr>
<td>total_swap_memory_free_bytes</td>
<td>2147479552</td>
</tr>
<tr>
<td>disk_space_free_mb</td>
<td>37102</td>
</tr>
<tr>
<td>disk_space_used_mb</td>
<td>11149</td>
</tr>
<tr>
<td>disk_space_total_mb</td>
<td>48251</td>
</tr>
</tbody>
</table>

- [ RECORD 3 ]----------------------------------------

<table>
<thead>
<tr>
<th>host_name</th>
<th>10.20.100.249</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_files_limit</td>
<td>65536</td>
</tr>
<tr>
<td>threads_limit</td>
<td>3833</td>
</tr>
<tr>
<td>core_file_limit_max_size_bytes</td>
<td>0</td>
</tr>
<tr>
<td>processor_count</td>
<td>2</td>
</tr>
<tr>
<td>processor_core_count</td>
<td>2</td>
</tr>
<tr>
<td>processor_description</td>
<td>Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz</td>
</tr>
<tr>
<td>opened_file_count</td>
<td>8</td>
</tr>
<tr>
<td>opened_socket_count</td>
<td>9</td>
</tr>
<tr>
<td>opened_nonfile_nonsocket_count</td>
<td>7</td>
</tr>
<tr>
<td>total_memory_bytes</td>
<td>4018823168</td>
</tr>
<tr>
<td>total_memory_free_bytes</td>
<td>241610752</td>
</tr>
<tr>
<td>total_buffer_memory_bytes</td>
<td>309395456</td>
</tr>
<tr>
<td>total_memory_cache_bytes</td>
<td>2881675264</td>
</tr>
<tr>
<td>total_swap_memory_bytes</td>
<td>2147479552</td>
</tr>
<tr>
<td>total_swap_memory_free_bytes</td>
<td>2147479552</td>
</tr>
<tr>
<td>disk_space_free_mb</td>
<td>37430</td>
</tr>
<tr>
<td>disk_space_used_mb</td>
<td>18821</td>
</tr>
<tr>
<td>disk_space_total_mb</td>
<td>48251</td>
</tr>
</tbody>
</table>

**IO_USAGE**

Provides disk I/O bandwidth usage history for the system.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Beginning of history interval.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>End of history interval.</td>
</tr>
<tr>
<td>READ_KBYTES_PER_SEC</td>
<td>FLOAT</td>
<td>Counter history of the number of bytes read measured in kilobytes per second.</td>
</tr>
<tr>
<td>WRITTEN_KBYTES_PER_SEC</td>
<td>FLOAT</td>
<td>Counter history of the number of bytes written measured in kilobytes per second.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**LDAP_LINK_EVENTS**

Monitors events that occurred during an LDAP Link synchronization.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The time the event occurred.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node or nodes for which the information is listed.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>The identification number of the LDAP Link session.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>The unique, system-generated user identification number.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user for which the information is listed.</td>
</tr>
</tbody>
</table>
| TRANSACTION_ID   | INTEGER     | The system-generated transaction identification number.  
Is NULL if a transaction id does not exist. |
<p>| EVENT_TYPE       | VARCHAR     | The type of event being logged, for example USER CREATED and PROCESSING_STARTED. |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY_NAME</td>
<td>VARCHAR</td>
<td>The name of the object on which the event occurred, if applicable. For example, the event SYNC-STARTED does not use an object.</td>
</tr>
<tr>
<td>ENTRY_OID</td>
<td>INTEGER</td>
<td>The unique identification number for the object on which the event occurred, if applicable.</td>
</tr>
<tr>
<td>LDAPURIHASH</td>
<td>INTEGER</td>
<td>The URI hash number for the LDAP user.</td>
</tr>
</tbody>
</table>

**LOAD_SOURCES**

Like LOAD_STREAMS, monitors active and historical load metrics on each node. The LOAD_SOURCES table breaks information down by source and portion. Rows appear in this table only for COPY operations that are profiled or run for more than one second. LOAD_SOURCES does not record information about loads from ORC or Parquet files or COPY LOCAL.

A row is added to this table when the loading of a source or portion begins. Column values related to the progress of the load are updated during the load operation.

The columns uniquely identifying the load source (the various ID and name columns) and IS_EXECUTING always have non-NULL values.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier of the session for which Vertica captures load stream information. This identifier is unique within the cluster for the current session but can be reused in a subsequent session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within a session. If a session is active, but no transaction has begun, this value is NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| STREAM_NAME  | VARCHAR   | Load stream identifier. If the user does not supply a specific name, the STREAM_NAME default value is `tablename-ID`, where:
|              |           | - `tablename` is the table into which data is being loaded.  
|              |           | - `ID` is an integer value. `ID` is guaranteed to be unique within the current session on a node.  
|              |           | This system table includes stream names for every COPY statement that takes more than 1 second to run. The 1-second duration includes the time to plan and execute the statement. |
| SCHEMA_NAME  | VARCHAR   | Schema name for which load information is listed. Lets you identify two streams that are targeted at tables with the same name in different schemas. NULL, if selecting from an external table. |
| TABLE_OID    | INTEGER   | A unique numeric ID assigned by the Vertica catalog that identifies the table. NULL, if selecting from an external table. |
| TABLE_NAME   | VARCHAR   | Name of the table being loaded. NULL, if selecting from an external table. |
| NODE_NAME    | VARCHAR   | Name of the node loading the source. |
| SOURCE_NAME  | VARCHAR   | - Full file path if copying from a file.  
|              |           | - Value returned by `getUri()` if the source is a user-defined source.  
<p>|              |           | - STDIN if loading from standard input. |
| PORTION_OFFSET | INTEGER | Offset of the source portion, or NULL if not apportioned. |
| PORTION_SIZE | INTEGER   | Size of the source portion, or NULL if not apportioned. |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Whether this source is currently being parsed, where t is true and f is false.</td>
</tr>
<tr>
<td>READ_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes read from the input file.</td>
</tr>
<tr>
<td>ROWS_PROroduced</td>
<td>INTEGER</td>
<td>Number of rows produced from parsing the source.</td>
</tr>
<tr>
<td>ROWS_REJECTED</td>
<td>INTEGER</td>
<td>Number of rows rejected from parsing the source. If CopyFaultTolerantExpressions is true, also includes rows rejected during expression evaluation.</td>
</tr>
<tr>
<td>INPUT_SIZE</td>
<td>INTEGER</td>
<td>Size of the input source in bytes, or NULL for unsized sources. For UDSources, this value is the value returned by getSize().</td>
</tr>
<tr>
<td>PARSE_COMPLETE_PERCENT</td>
<td>INTEGER</td>
<td>Percent of rows from the input file that have been parsed.</td>
</tr>
<tr>
<td>FAILURE_REASON</td>
<td>VARCHAR</td>
<td>For load failures, error message indicating why loading from this source failed or was cancelled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• In cases of a node failure, exception, or cancellation, this column contains the associated error message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If ERROR TOLERANCE was not used and a failure occurred, &quot;Source aborted due to exception while loading from (name)&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For any other failure, &quot;Exception occurred during load&quot;.</td>
</tr>
<tr>
<td>PEAK_COOPERATING_THREADS</td>
<td>INTEGER</td>
<td>The peak number of threads parsing this source in parallel.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.
Examples

The following example shows the values you would see in the LOAD_SOURCES table after some loads with various errors.

```sql
=> CREATE TABLE tt (i int);
=> COPY tt FROM '/tmp/load_sources_large.dat';
^CCancel request sent
ERROR 3322: Execution canceled by operator

=> COPY tt FROM '/tmp/load_sources_large.dat' ON ANY NODE;
  Rows Loaded
     ----------
     1000000
(1 row)

=> COPY tt FROM '/tmp/load_sources_*._txt' ON ANY NODE;
ERROR 2018: COPY: Could not open file [/tmp/load_sources_perm.txt] for reading; Permission denied
HINT: Make sure the file is available on the specified node. If using ON ANY NODE, it must be available at this path on all nodes in the cluster since any node could load the file

=> COPY tt FROM '/tmp/load_sources_*._txt' ON ANY NODE ERROR TOLERANCE;
WARNING 7177: Error loading from 'load_sources_perm.txt' on node 'e1'
DETAIL: ERROR 2018: COPY: Could not open file [/tmp/load_sources_perm.txt] for reading; Permission denied
WARNING 7176: Error loading from 1/4 sources
  Rows Loaded
     ----------
     3000
(1 row)

=> SELECT * FROM LOAD_SOURCES;

<table>
<thead>
<tr>
<th>session_id</th>
<th>transaction_id</th>
<th>statement_id</th>
<th>stream_name</th>
<th>schema_name</th>
<th>table_oid</th>
<th>table_name</th>
<th>node_name</th>
<th>source_name</th>
<th>input_size</th>
<th>portion_offset</th>
<th>portion_size</th>
<th>read_bytes</th>
<th>parsed_bytes</th>
<th>rows_produced</th>
<th>rowsRejected</th>
<th>parse_complete_percent</th>
<th>isExecuting</th>
<th>failure_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273705528</td>
<td>4503599627374343</td>
<td>tt</td>
<td>initiator</td>
<td>/tmp/load_sources_large.dat</td>
<td>1354168</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1354168</td>
<td>1048576</td>
<td>489777</td>
<td>53310</td>
<td>26655</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| f | ERROR 3322: Execution canceled by operator

| initiator-9959:0x68 | 45035996273705528 | 3 | | public | 4503599627374343 | tt | initiator | /tmp/load_sources_large.dat | 1354168 | 3 | | | | | | | | | | |
| 541671 | 1354168 | 1048576 | 542407 | 49310 | 24655 | 48 | f | ERROR 3322: Execution canceled by operator

| initiator-9959:0x68 | 45035996273705528 | 3 | | public | 4503599627374343 | tt | initiator | /tmp/load_sources_large.dat | 1354168 | 3 | | | | | | | | | | |
| 2708336 | 1354168 | 1048576 | 541838 | 49258 | 24629 | 48 | f | ERROR 3322: Execution canceled by operator

| initiator-9959:0x68 | 45035996273705528 | 3 | | public | 4503599627374343 | tt | initiator | /tmp/load_sources_large.dat | 1354168 | 3 | | | | | | | | | | |
| 8125006 | 1354168 | 1048576 | 561390 | 51036 | 25518 | | | | | | | | | | | | | |
```
| 41 | f | ERROR 3322: Execution canceled by operator |
| initiator-9959:0x68 | 45035996273705528 | 3 | public |
| 45035996273743434 | tt | initiator | /tmp/load_sources_large.dat | 1354168 |
| 1354168 | 1354168 | 1048576 | 555788 | 50526 | 25264 |
| 41 | f | ERROR 3322: Execution canceled by operator |
| initiator-9959:0x68 | 45035996273705528 | 3 | public |
| 45035996273743434 | tt | initiator | /tmp/load_sources_large.dat | 1354167 |
| 6770839 | 1354167 | 1048576 | 549467 | 49952 | 24976 |
| 40 | f | ERROR 3322: Execution canceled by operator |
| initiator-9959:0x68 | 45035996273705528 | 3 | public |
| 45035996273743434 | tt | initiator | /tmp/load_sources_large.dat | 1354167 |
| 9479174 | 1354167 | 1048576 | 569858 | 51805 | 25903 |
| 42 | f | ERROR 3322: Execution canceled by operator |
| initiator-9959:0x68 | 45035996273705528 | 3 | public |
| 45035996273743434 | tt | initiator | /tmp/load_sources_large.dat | 1354167 |
| 4062804 | 1354167 | 1048576 | 547468 | 49770 | 24885 |
| 40 | f | ERROR 3322: Execution canceled by operator |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e0 | /tmp/load_sources_large.dat | 1805557 |
| 3611115 | 1805557 | 2097152 | 1805561 | 164141 | 82071 |
| 100 | f | public |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e0 | /tmp/load_sources_large.dat | 1805556 |
| 5416672 | 1805556 | 2097152 | 1805558 | 164142 | 82070 |
| 100 | f | public |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_large.dat | 1805558 |
| 0 | 1805558 | 2097152 | 1805559 | 179293 | 89646 | 100 |
| f | public |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_large.dat | 1805557 |
| 1805558 | 1805557 | 2097152 | 1805563 | 164142 | 82071 |
| 100 | f | public |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_large.dat | 1805557 |
| 7222228 | 1805557 | 2097152 | 1805564 | 164142 | 82071 |
| 100 | f | public |
| initiator-9959:0x68 | 45035996273705528 | 4 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_large.dat | 1805556 |
| 9827785 | 1805556 | 1805556 | 1805556 | 164140 | 82071 |
| 100 | f | public |
| initiator-9959:0x68 | 45035996273705528 | 1 | public |
| 45035996273743434 | tt | e0 | /tmp/load_sources_2.txt | 7838 |
| | 7838 | 553 | 88 | 44 | 7 |
| f | Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on node 'e1'. |
| initiator-9959:0x68 | 45035996273705532 | 1 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_1.txt | 7838 |
| | 7838 | 215 | 36 | 18 | 2 |
| f | Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on node 'e1'. |
| initiator-9959:0x68 | 45035996273705532 | 1 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_perm.txt | |
| | 0 | 0 | 0 | 0 | 0 |
| f | ERROR 2818: COPY: Could not open file ['/tmp/load_sources_perm.txt'] for reading; Permission denied |
| initiator-9959:0x68 | 45035996273705532 | 1 | public |
| 45035996273743434 | tt | e1 | /tmp/load_sources_3.txt | 7838 |
| | 7838 | 2327 | 312 | 156 | 29 |
| f | Source aborted due to exception while loading from '/tmp/load_sources_perm.txt' on
LOAD_STREAMS

Monitors active and historical load metrics for load streams. This is useful for obtaining statistics about how many records got loaded and rejected from the previous load. Vertica maintains system table metrics until they reach a designated size quota (in kilobytes). This quota is set through internal processes, which you cannot set or view directly.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier of the session for which Vertica captures load stream information. This identifier is unique within the cluster for the current session, but can be reused in a subsequent session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within a session. If a session is active but no transaction has begun, this is NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
</tbody>
</table>
| STREAM_NAME    | VARCHAR   | Load stream identifier. If the user does not supply a
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>specific name, the STREAM_NAME default value is: tablename-ID where tablename is the table into which data is being loaded, and ID is an integer value, guaranteed to be unique with the current session on a node. This system table includes stream names for every COPY statement that takes more than 1-second to run. The 1-second duration includes the time to plan and execute the statement.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>Schema name for which load stream information is listed. Lets you identify two streams that are targeted at tables with the same name in different schemas</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the table being loaded.</td>
</tr>
<tr>
<td>LOAD_START</td>
<td>VARCHAR</td>
<td>Linux system time when the load started.</td>
</tr>
<tr>
<td>LOAD_DURATION_MS</td>
<td>NUMERIC (54,0)</td>
<td>Duration of the load stream in milliseconds.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Indicates whether the load is executing, where t is true and f is false.</td>
</tr>
<tr>
<td>ACCEPTED_ROW_COUNT</td>
<td>INTEGER</td>
<td>Number of rows loaded.</td>
</tr>
<tr>
<td>REJECTED_ROW_COUNT</td>
<td>INTEGER</td>
<td>Number of rows rejected.</td>
</tr>
<tr>
<td>READBYTES</td>
<td>INTEGER</td>
<td>Number of bytes read from the input file.</td>
</tr>
<tr>
<td>INPUT_FILE_SIZE_BYTES</td>
<td>INTEGER</td>
<td>Size of the input file in bytes. Note: When using STDIN as input, the input file size is zero (0).</td>
</tr>
<tr>
<td>PARSE_COMPLETE_PERCENT</td>
<td>INTEGER</td>
<td>Percent of rows from the input file that have been parsed.</td>
</tr>
<tr>
<td>UNSORTED_ROW_COUNT</td>
<td>INTEGER</td>
<td>Cumulative number rows not sorted across all</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>projections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> UNSORTED_ROW_COUNT could be greater than ACCEPTED_ROW_COUNT because data is copied and sorted for every projection in the target table.</td>
</tr>
<tr>
<td>SORTED_ROW_COUNT</td>
<td>INTEGER</td>
<td>Cumulative number of rows sorted across all projections.</td>
</tr>
<tr>
<td>SORT_COMPLETE_PERCENT</td>
<td>INTEGER</td>
<td>Percent of rows from the input file that have been sorted.</td>
</tr>
</tbody>
</table>

**Privileges**

If you have the SYSMONITOR role or are the dbadmin user, this table shows all loads. Otherwise it shows only your loads.

**LOCK_USAGE**

Provides aggregate information about lock requests, releases, and attempts, such as wait time/count and hold time/count. Vertica records:

- Lock attempts at the end of the locking process
- Lock releases after lock attempts are released

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information on which lock interaction occurs.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>Name of object being locked; can be a table or an internal structure (projection, global catalog, or local catalog).</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MODE</td>
<td>VARCHAR</td>
<td>Intended operations of the transaction. Otherwise, this value is NONE. For a list of lock modes and compatibility, see Lock Modes.</td>
</tr>
<tr>
<td>AVG_HOLD_TIME</td>
<td>INTERVAL</td>
<td>Average time (measured in intervals) that Vertica holds a lock.</td>
</tr>
<tr>
<td>MAX_HOLD_TIME</td>
<td>INTERVAL</td>
<td>Maximum time (measured in intervals) that Vertica holds a lock.</td>
</tr>
<tr>
<td>HOLD_COUNT</td>
<td>INTEGER</td>
<td>Total number of times the lock was granted in the given mode.</td>
</tr>
<tr>
<td>AVG_WAIT_TIME</td>
<td>INTERVAL</td>
<td>Average time (measured in intervals) that Vertica waits on the lock.</td>
</tr>
<tr>
<td>MAX_WAIT_TIME</td>
<td>INTERVAL</td>
<td>Maximum time (measured in intervals) that Vertica waits on a lock.</td>
</tr>
<tr>
<td>WAIT_COUNT</td>
<td>INTEGER</td>
<td>Total number of times lock was unavailable at the time it was first requested.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**See Also**

- DUMP_LOCKTABLE
- LOCKS
- PROJECTION_REFRESHES
- SELECT
- SESSION_PROFILES
# LOCKS

Monitors lock grants and requests for all nodes. A table call with no results indicates that no locks are in use.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAMES</td>
<td>VARCHAR</td>
<td>Nodes on which lock interaction occurs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Node Rollup:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NODE_NAMES are separated by commas. A transaction can have the same lock in the same mode in the same scope on multiple nodes. However, the transaction gets only one (1) line in the table.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>Name of object being locked; can be a table or an internal structure (projection, global catalog, or local catalog).</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog that identifies the object being locked.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>VARCHAR</td>
<td>Identification of transaction within the session, if any; otherwise NULL. Useful for creating joins to other system tables.</td>
</tr>
<tr>
<td>TRANSACTION_DESCRIPTION</td>
<td>VARCHAR</td>
<td>Identification of transaction and associated description. Typically this query caused the transaction's creation.</td>
</tr>
<tr>
<td>LOCK_MODE</td>
<td>VARCHAR</td>
<td>Intended operation of the transaction. For a list of lock modes and compatibility, see <a href="#">Lock Modes</a>.</td>
</tr>
</tbody>
</table>
| LOCK_SCOPE     | VARCHAR     | Expected duration of the lock after it is granted. Before the lock is granted, Vertica lists the scope as REQUESTED. Once a lock has been granted, the following scopes are possible:  
  - STATEMENT_LOCALPLAN |
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT_COMPILE</td>
<td></td>
<td>• STATEMENT_COMPILE</td>
</tr>
<tr>
<td>STATEMENT_EXECUTE</td>
<td></td>
<td>• STATEMENT_EXECUTE</td>
</tr>
<tr>
<td>TRANSACTION_POSTCOMMIT</td>
<td></td>
<td>• TRANSACTION_POSTCOMMIT</td>
</tr>
<tr>
<td>TRANSACTION</td>
<td></td>
<td>• TRANSACTION</td>
</tr>
</tbody>
</table>

All scopes, other than TRANSACTION, are transient and are used only as part of normal query processing.

<table>
<thead>
<tr>
<th>REQUEST_TIMESTAMP</th>
<th>TIMESTAMP</th>
<th>Time when the transaction began waiting on the lock.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANT_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Time the transaction acquired or upgraded the lock:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Return values are NULL until the grant occurs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If the grant occurs immediately, values might be the same as REQUEST_TIMESTAMP.</td>
</tr>
</tbody>
</table>

See Also

- About Vertica Locks
- DUMP_LOCKTABLE
- LOCK_USAGE
- PROJECTION_REFRESHES
- SELECT
- SESSION_PROFILES
- TRANSACTIONS
LOGIN_FAILURES

This system table lists failures for each failed login attempt. This information helps you determine if a user is having difficulty getting into the database or identify a possible intrusion attempt.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGIN_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica recorded the login.</td>
</tr>
<tr>
<td>DATABASE_NAME</td>
<td>VARCHAR</td>
<td>The name of the database for the login attempt.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user whose login failed at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>CLIENT_HOSTNAME</td>
<td>VARCHAR</td>
<td>Host name and port of the TCP socket from which the client connection was made. NULL if the session is internal.</td>
</tr>
<tr>
<td>CLIENT_PID</td>
<td>INTEGER</td>
<td>Identifier of the client process that issued this connection. In some cases, the client process is on a different machine from the server.</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td>Unused.</td>
</tr>
<tr>
<td>CLIENT_OS_USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.</td>
</tr>
<tr>
<td>AUTHENTICATION_METHOD</td>
<td>VARCHAR</td>
<td>Name of the authentication method used to validate the client application or user who is trying to connect to the server using the database user name provided</td>
</tr>
</tbody>
</table>

**Valid values:**

- Trust
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Reject</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LDAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ident</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hash</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TLS</td>
</tr>
<tr>
<td>See <a href="#">Implementing Client Authentication</a> in the Administrator's Guide for further information.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLIENT_AUTHENTICATION_NAME</th>
<th>VARCHAR</th>
<th>Locally created name of the client authentication method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>REASON</td>
<td>VARCHAR</td>
<td>Description of login failure reason. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INVALID USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ACCOUNT LOCKED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REJECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FAILED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INVALID AUTH METHOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INVALID DATABASE</td>
</tr>
</tbody>
</table>

### Privileges

Superuser
MEMORY_USAGE

Records system resource history for memory usage. This is useful for comparing memory that Vertica uses versus memory in use by the entire system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Beginning of history interval.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>End of history interval.</td>
</tr>
<tr>
<td>AVERAGE_MEMORY_USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Records the average memory usage in percent of total memory (0-100) during the history interval.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

MONITORING_EVENTS

Reports significant events that can affect database performance and functionality if you do not address their root causes.

See Monitoring Events in the Administrator's Guide for details.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>EVENT_CODE</td>
<td>INTEGER</td>
<td>Numeric identifier that indicates the type of event. See Event Types in Monitoring Events in the Administrator's Guide for a list of event type codes.</td>
</tr>
<tr>
<td>EVENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID that identifies the specific event.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| EVENT_SEVERITY              | VARCHAR     | Severity of the event from highest to lowest. These events are based on standard syslog severity types:  
0  – Emergency  
1  – Alert  
2  – Critical  
3  – Error  
4  – Warning  
5  – Notice  
6  – Info  
7  – Debug |
| EVENT_POSTED_TIMESTAMP      | TIMESTAMPTZ | When this event was posted.                                                |
| EVENT_CLEARED_TIMESTAMP     | TIMESTAMPTZ | When this event was cleared.                                               |
|                             |             | **Note:** You can also query the ACTIVE_EVENTS system table to see events that have not been cleared. |
| EVENT_EXPIRATION            | TIMESTAMPTZ | Time at which this event expires. If the same event is posted again prior to its expiration time, this field gets updated to a new expiration time. |
| EVENT_CODE_DESCRIPTION      | VARCHAR     | Brief description of the event and details pertinent to the specific situation. |
| EVENT_PROBLEM_DESCRIPTION   | VARCHAR     | Generic description of the event.                                           |

**Privileges**

Superuser

**See Also**

ACTIVE_EVENTS
NETWORK_INTERFACES

Provides information about network interfaces on all Vertica nodes.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>Unique identifier for the node that recorded the row.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>VARCHAR</td>
<td>Network interface name.</td>
</tr>
<tr>
<td>IP_ADDRESS_FAMILY</td>
<td>VARCHAR</td>
<td>Network address protocol.</td>
</tr>
<tr>
<td>IP_ADDRESS</td>
<td>VARCHAR</td>
<td>IP address for this interface.</td>
</tr>
<tr>
<td>SUBNET</td>
<td>VARCHAR</td>
<td>IP subnet for this interface.</td>
</tr>
<tr>
<td>MASK</td>
<td>VARCHAR</td>
<td>IP network mask for this interface.</td>
</tr>
<tr>
<td>BROADCAST_ADDRESS</td>
<td>VARCHAR</td>
<td>IP broadcast address for this interface.</td>
</tr>
</tbody>
</table>

Privileges

None

NETWORK_USAGE

Provides network bandwidth usage history on the system. This is useful for determining if Vertica is using a large percentage of its available network bandwidth.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Beginning of history interval.</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
END_TIME | TIMESTAMP | End of history interval.
TX_KBYTES_PER_SEC | FLOAT | Counter history of outgoing (transmitting) usage in kilobytes per second.
RX_KBYTES_PER_SEC | FLOAT | Counter history of incoming (receiving) usage in kilobytes per second.

### Privileges
Superuser

### NODE_EVICTIONS
Monitors node evictions on the system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVICTION_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Timestamp when the eviction request was made.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name logging the information.</td>
</tr>
<tr>
<td>EVICTED_NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name of the evicted node.</td>
</tr>
<tr>
<td>EVICTED_NODE_ID</td>
<td>INTEGER</td>
<td>The evicted node ID.</td>
</tr>
<tr>
<td>NODE_STATE_BEFORE_EVICTION</td>
<td>VARCHAR</td>
<td>The previous node state at the time of eviction.</td>
</tr>
</tbody>
</table>

### NODE_RESOURCES
Provides a snapshot of the node. This is useful for regularly polling the node with automated tools or scripts.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HOST_NAME</td>
<td>VARCHAR</td>
<td>The hostname associated with a particular node.</td>
</tr>
<tr>
<td>PROCESS_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The total size of the program.</td>
</tr>
<tr>
<td>PROCESS_RESIDENT_SET_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The total number of bytes that the process has in memory.</td>
</tr>
<tr>
<td>PROCESS_SHARED_MEMORY_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The amount of shared memory used.</td>
</tr>
<tr>
<td>PROCESS_TEXT_MEMORY_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The total number of text bytes that the process has in physical memory. This does not include any shared libraries.</td>
</tr>
<tr>
<td>PROCESS_DATA_MEMORY_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The amount of physical memory, in bytes, used for performing processes. This does not include the executable code.</td>
</tr>
<tr>
<td>PROCESS_LIBRARY_MEMORY_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The total number of library bytes that the process has in physical memory.</td>
</tr>
<tr>
<td>PROCESS_DIRTY_MEMORY_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes that have been modified since they were last written to disk.</td>
</tr>
<tr>
<td>SPREAD_HOST</td>
<td>VARCHAR</td>
<td>The node name of the spread host.</td>
</tr>
<tr>
<td>NODE_PORT</td>
<td>VARCHAR</td>
<td>The port used for intra-cluster communication.</td>
</tr>
<tr>
<td>DATA_PORT</td>
<td>VARCHAR</td>
<td>The port used by the Vertica client.</td>
</tr>
</tbody>
</table>

**NODE_STATES**

Monitors node recovery state-change history on the system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica recorded the event.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>NODE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the node.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>NODE_STATE</td>
<td>VARCHAR</td>
<td>Shows the node's state. Can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UNSAFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SHUTDOWN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SHUTDOWN ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RECOVERING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RECOVERY ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RECOVERED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INITIALIZING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• STAND BY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NEEDS CATCH UP</td>
</tr>
</tbody>
</table>

The following flow chart details different node states:
Privileges
None

NODE_SUBSCRIPTIONS

Lists the node subscriptions in the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSCRIPTION_OID</td>
<td>INTEGER</td>
<td>The OID of the subscription.</td>
</tr>
<tr>
<td>NODE_OID</td>
<td>INTEGER</td>
<td>The OID of the node subscribed.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node.</td>
</tr>
<tr>
<td>SHARD_OID</td>
<td>INTEGER</td>
<td>The OID of the shard to which the node is subscribed.</td>
</tr>
<tr>
<td>SHARD_NAME</td>
<td>VARCHAR</td>
<td>The name of the shard to which the node is subscribed.</td>
</tr>
<tr>
<td>SUBSCRIPTION_STATE</td>
<td>VARCHAR</td>
<td>The current subscription state of the node.</td>
</tr>
<tr>
<td>IS_PRIMARY</td>
<td>BOOLEAN</td>
<td>Defines whether the node is currently the primary subscriber.</td>
</tr>
<tr>
<td>IS_RESUBSCRIBING</td>
<td>BOOLEAN</td>
<td>Indicates whether a subscription is resubscribing to a node, as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t (true) - a subscription is resubscribing, only applies to PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subscriptions created during the cluster/node startup.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f (false) - a subscription is not resubscribing, applies to PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subscriptions created with REBALANCE_SHARDS that have transitioned to an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACTIVE state.</td>
</tr>
<tr>
<td>HAS_CURRENT_METADATA</td>
<td>BOOLEAN</td>
<td>Indicates whether or not the subscription contains the most recent metadata.</td>
</tr>
<tr>
<td>DEPOT_WARM</td>
<td>BOOLEAN</td>
<td>Indicates whether or not the subscription contains the most recent depot updates.</td>
</tr>
</tbody>
</table>
NOTIFIER_ERRORS

Reports errors encountered by notifiers.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_TIME</td>
<td>TIMESTAMPZ</td>
<td>The time that the error occurred.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that encountered the error.</td>
</tr>
<tr>
<td>NOTIFIER_NAME</td>
<td>VARCHAR</td>
<td>Name of the notifier that triggered the error.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>A description of the error.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

OUTPUT_DEPLOYMENT_STATUS

Contains information about the deployment status of all the projections in your design. Each row contains information about a different projection. Vertica populates this table when you deploy the database design by running the function DESIGNER_RUN_POPULATE DESIGN AND DEPLOY.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment_id</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the deployment.</td>
</tr>
<tr>
<td>design_name</td>
<td>VARCHAR</td>
<td>Unique name that the user assigned to the design.</td>
</tr>
<tr>
<td>deployment_projection_id</td>
<td>INTEGER</td>
<td>Unique ID that Database Designer assigned to the output projection.</td>
</tr>
<tr>
<td>deployment_projection_name</td>
<td>VARCHAR</td>
<td>Name that Database Designer assigned to the output projection or the name of the projection to be</td>
</tr>
</tbody>
</table>
### Column Name | Column Type | Description
---|---|---
name | | dropped.

**deployment_status** | VARCHAR | Status of the deployment:

- pending
- complete
- needs_refresh
- in_progress
- error

**error_message** | VARCHAR | Text of any error that occurred when creating or refreshing the specified projection.

---

**OUTPUT_EVENT_HISTORY**

Contains information about each stage that Database Designer performs to design and optimize your database design.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_STAMP</td>
<td>TIMESTAMP</td>
<td>Date and time of the specified stage.</td>
</tr>
<tr>
<td>DESIGN_ID</td>
<td>INTEGER</td>
<td>Unique id that Database Designer assigned to the design.</td>
</tr>
<tr>
<td>DESIGN_NAME</td>
<td>VARCHAR</td>
<td>Unique name that the user assigned to the design.</td>
</tr>
</tbody>
</table>
| STAGE_TYPE | VARCHAR | Design stage that Database Designer was working on at the time indicated by the TIME_STAMP field. Possible values include:

- Design in progress
- Analyzing data statistics
- Optimizing query performance |
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Optimizing storage footprint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• All done</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Deployment in progress</td>
</tr>
<tr>
<td>ITERATION_NUMBER</td>
<td>INTEGER</td>
<td>Iteration number for the Optimizing query performance stage.</td>
</tr>
<tr>
<td>TOTAL_QUERY_COUNT</td>
<td>INTEGER</td>
<td>Total number of design queries in the design.</td>
</tr>
<tr>
<td>REMAINING_QUERY_COUNT</td>
<td>INTEGER</td>
<td>Number of design queries remaining for Database Designer to process.</td>
</tr>
<tr>
<td>MAX_STEP_NUMBER</td>
<td>INTEGER</td>
<td>Number of steps in the current stage.</td>
</tr>
<tr>
<td>CURRENT_STEP_NUMBER</td>
<td>INTEGER</td>
<td>Step in the current stage being processed at the time indicated by the TIME_STAMP field.</td>
</tr>
<tr>
<td>CURRENT_STEP_DESCRIPTION</td>
<td>VARCHAR</td>
<td>Name of the step that Database Designer is performing at that time indicated in the TIME_STAMP field. Possible values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design with deployment started</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design in progress: Analyze statistics phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• design_table_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• projection_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design in progress: Query optimization phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Extracting interesting columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enumerating sort orders</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Setting up projection candidates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Assessing projection candidates</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Choosing best projections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Calculating estimated benefit of best projections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Design in progress: Storage optimization phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Design completed successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Setting up deployment metadata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Identifying projections to be dropped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Running deployment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Deployment completed successfully</td>
</tr>
</tbody>
</table>

| TABLE_ID   | INTEGER    | Unique id that Database Designer assigned to the design table. |

Example

The following example shows the steps that Database Designer performs while optimizing the VMart example database:

```sql
=> SELECT DESIGNER_CREATE_DESIGN('VMART_DESIGN');
=> SELECT DESIGNER_ADD DESIGN TABLES('VMART_DESIGN','public.*');
=> SELECT DESIGNER_ADD DESIGN QUERIES('VMART_DESIGN','/tmp/examples/vmart_queries.sql');;
...
=> \x
Expanded display is on.
=> SELECT * FROM OUTPUT_EVENT_HISTORY;
```

<table>
<thead>
<tr>
<th>time_stamp</th>
<th>2013-06-05 11:44:41.588</th>
</tr>
</thead>
<tbody>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>VMART_DESIGN</td>
</tr>
<tr>
<td>stage_type</td>
<td>Design in progress</td>
</tr>
<tr>
<td>iteration_number</td>
<td></td>
</tr>
<tr>
<td>total_query_count</td>
<td></td>
</tr>
<tr>
<td>remaining_query_count</td>
<td></td>
</tr>
<tr>
<td>max_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_description</td>
<td>Design with deployment started</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>table id</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>RECORD 2</td>
<td></td>
</tr>
<tr>
<td>time_stamp</td>
<td>2013-06-05 11:44:41.611</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>VMART_DESIGN</td>
</tr>
<tr>
<td>stage_type</td>
<td>Design in progress</td>
</tr>
<tr>
<td>iteration_number</td>
<td></td>
</tr>
<tr>
<td>total_query_count</td>
<td></td>
</tr>
<tr>
<td>remaining_query_count</td>
<td></td>
</tr>
<tr>
<td>max_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_description</td>
<td>Design in progress: Analyze statistics phase</td>
</tr>
<tr>
<td>table id</td>
<td></td>
</tr>
<tr>
<td>RECORD 3</td>
<td></td>
</tr>
<tr>
<td>time_stamp</td>
<td>2013-06-05 11:44:42.011</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>VMART_DESIGN</td>
</tr>
<tr>
<td>stage_type</td>
<td>Analyzing statistics</td>
</tr>
<tr>
<td>iteration_number</td>
<td></td>
</tr>
<tr>
<td>total_query_count</td>
<td></td>
</tr>
<tr>
<td>remaining_query_count</td>
<td></td>
</tr>
<tr>
<td>max_step_number</td>
<td>15</td>
</tr>
<tr>
<td>current_step_number</td>
<td>1</td>
</tr>
<tr>
<td>current_step_description</td>
<td>public.customer_dimension</td>
</tr>
<tr>
<td>table id</td>
<td></td>
</tr>
<tr>
<td>RECORD 20</td>
<td></td>
</tr>
<tr>
<td>time_stamp</td>
<td>2013-06-05 11:44:49.324</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>VMART_DESIGN</td>
</tr>
<tr>
<td>stage_type</td>
<td>Optimizing query performance</td>
</tr>
<tr>
<td>iteration_number</td>
<td>1</td>
</tr>
<tr>
<td>total_query_count</td>
<td>9</td>
</tr>
<tr>
<td>remaining_query_count</td>
<td>9</td>
</tr>
<tr>
<td>max_step_number</td>
<td>7</td>
</tr>
<tr>
<td>current_step_number</td>
<td>1</td>
</tr>
<tr>
<td>current_step_description</td>
<td>Extracting interesting columns</td>
</tr>
<tr>
<td>table id</td>
<td></td>
</tr>
<tr>
<td>RECORD 62</td>
<td></td>
</tr>
<tr>
<td>time_stamp</td>
<td>2013-06-05 11:51:23.790</td>
</tr>
<tr>
<td>design_id</td>
<td>45035996273705090</td>
</tr>
<tr>
<td>design_name</td>
<td>VMART_DESIGN</td>
</tr>
<tr>
<td>stage_type</td>
<td>Deployment in progress</td>
</tr>
<tr>
<td>iteration_number</td>
<td></td>
</tr>
<tr>
<td>total_query_count</td>
<td></td>
</tr>
<tr>
<td>remaining_query_count</td>
<td></td>
</tr>
<tr>
<td>max_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_number</td>
<td></td>
</tr>
<tr>
<td>current_step_description</td>
<td>Deployment completed successfully</td>
</tr>
<tr>
<td>table id</td>
<td></td>
</tr>
</tbody>
</table>
PARTITION_COLUMNS

For each projection of each partitioned table, shows the disk space used by each column on each node. The column disk_space_bytes shows how much disk space the partitioned data uses, including deleted data. So, if you delete rows but do not not purge them, the view's deleted_row_count column changes to show the number of deleted rows in each column; however, disk_space_bytes remains the same. After the deleted rows are purged, Vertica, reclaims the disk space: disk_space_bytes changes accordingly, and deleted_row_count is reset to 0.

For grouped partitions, PARTITION_COLUMNS shows the cumulative disk space used for each column per grouped partition. The column grouped_partition_key, if not null, identifies the partition in which a given column is grouped.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR</td>
<td>Identifies a named column within the partitioned table.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica, which identifies the column.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the partitioned table.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Projection name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by Vertica, which identifies the projection.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node that hosts partitioned data.</td>
</tr>
<tr>
<td>PARTITION_KEY</td>
<td>VARCHAR</td>
<td>Identifies the table partition.</td>
</tr>
<tr>
<td>GROUPED_PARTITION_KEY</td>
<td>VARCHAR</td>
<td>Identifies the grouped partition to which a given column belongs.</td>
</tr>
<tr>
<td>ROW_COUNT</td>
<td>INTEGER</td>
<td>The total number of partitioned data rows for each column, including deleted rows.</td>
</tr>
<tr>
<td>DELETED_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of deleted partitioned data rows in each column.</td>
</tr>
</tbody>
</table>
Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

Example

Given the following table definition:

```sql
=> CREATE TABLE messages
(    time_interval timestamp NOT NULL,
    thread_id varchar(32) NOT NULL,
    unique_id varchar(53) NOT NULL,
    msg_id varchar(65),
    ...)
PARTITION BY ((messages.time_interval)::date);
```

A query on partition_columns might return the following (truncated) results:

```sql
=> SELECT * FROM partition_columns order by table_name, column_name;
```

<table>
<thead>
<tr>
<th>column_name</th>
<th>column_id</th>
<th>table_name</th>
<th>projection_name</th>
<th>projection_id</th>
<th>node_name</th>
<th>partition_key</th>
<th>grouped_partition_key</th>
<th>row_count</th>
<th>deleted_row_count</th>
<th>disk_space_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>msg_id</td>
<td>45035996273743190</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0002</td>
<td>2010-07-03</td>
<td>6147</td>
<td>0</td>
<td>41145</td>
<td></td>
</tr>
<tr>
<td>msg_id</td>
<td>45035996273743190</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0002</td>
<td>2010-07-15</td>
<td>178</td>
<td>0</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>msg_id</td>
<td>45035996273743190</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0003</td>
<td>2010-07-03</td>
<td>6782</td>
<td>0</td>
<td>45107</td>
<td></td>
</tr>
<tr>
<td>msg_id</td>
<td>45035996273743190</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0003</td>
<td>2010-07-04</td>
<td>866</td>
<td>0</td>
<td>5883</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread_id</td>
<td>45035996273743186</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0002</td>
<td>2010-07-03</td>
<td>6147</td>
<td>0</td>
<td>78565</td>
<td></td>
</tr>
<tr>
<td>thread_id</td>
<td>45035996273743186</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0002</td>
<td>2010-07-15</td>
<td>178</td>
<td>0</td>
<td>2429</td>
<td></td>
</tr>
<tr>
<td>thread_id</td>
<td>45035996273743186</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0003</td>
<td>2010-07-03</td>
<td>6782</td>
<td>0</td>
<td>77730</td>
<td></td>
</tr>
<tr>
<td>thread_id</td>
<td>45035996273743186</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0003</td>
<td>2010-07-04</td>
<td>866</td>
<td>0</td>
<td>10317</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time_interval</td>
<td>45035996273743184</td>
<td>messages</td>
<td>messages_super</td>
<td>45035996273743182</td>
<td>v_vmart_node0002</td>
<td>2010-07-03</td>
<td>6147</td>
<td>0</td>
<td>78565</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
| 2010-07-03 | 6147 | 0 | 6320 | time_interval | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0002 |
| 2010-07-15 | 178 | 0 | 265 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
| 2010-07-03 | 6782 | 0 | 6967 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
| 2010-07-04 | 866 | 0 | 892 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |

... unique_id | 45035996273743188 | messages | messages_super | 45035996273743182 | v_vmart_node0002 |
| 2010-07-03 | 6147 | 0 | 70747 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0002 |
| 2010-07-15 | 178 | 0 | 2460 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
| 2010-07-03 | 6782 | 0 | 77959 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
| 2010-07-04 | 866 | 0 | 10332 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |
| 2010-07-15 | 184 | 0 | 2549 | | 45035996273743184 | messages | messages_super | 45035996273743182 | v_vmart_node0003 |

(11747 rows)

**PARTITION_REORGANIZE_ERRORS**

new column `projection_id`

Monitors all background partitioning tasks, and if Vertica encounters an error, creates an entry in this table with the appropriate information. Does not log repartitioning tasks that complete successfully.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user who received the error at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the partitioned table.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Projection name for which information is listed.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>VARCHAR</td>
<td>Textual output of the error message.</td>
</tr>
<tr>
<td>HINT</td>
<td>VARCHAR</td>
<td>Actionable hint about the error.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**PARTITION_STATUS**

For each projection of each partitioned table, shows the fraction of its data that is actually partitioned according to the current partition expression. When the partitioning of a table is altered, the value in PARTITION_REORGANIZE_PERCENT for each of its projections drops to zero and goes back up to 100 when all the data is repartitioned.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the schema that contains the partitioned table.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Table name that is partitioned.</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica, which identifies the table.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema containing the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Projection name for which information is listed.</td>
</tr>
<tr>
<td>PARTITION_REORGANIZE_PERCENT</td>
<td>INTEGER</td>
<td>For each projection, drops to zero and goes back up to 100 when all the data is repartitioned after the partitioning of a table has been altered. Ideally all rows will show 100 (%).</td>
</tr>
</tbody>
</table>
Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

PARTITIONS

Displays partition metadata, one row per partition key, per ROS container.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION_KEY</td>
<td>VARCHAR</td>
<td>The partition value(s).</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed.</td>
</tr>
<tr>
<td>ROS_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the ROS container.</td>
</tr>
<tr>
<td>ROS_SIZE_BYTES</td>
<td>INTEGER</td>
<td>The ROS container size in bytes.</td>
</tr>
<tr>
<td>ROS_ROW_COUNT</td>
<td>INTEGER</td>
<td>Number of rows in the ROS container.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node where the ROS container resides.</td>
</tr>
<tr>
<td>DELETED_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in the partition.</td>
</tr>
<tr>
<td>LOCATION_LABEL</td>
<td>VARCHAR</td>
<td>The location label of the default storage location.</td>
</tr>
</tbody>
</table>

Notes

- A many-to-many relationship exists between partitions and ROS containers. PARTITIONS displays information in a denormalized fashion.
To find the number of ROS containers having data of a specific partition, aggregate PARTITIONS over the partition_key column.

To find the number of partitions stored in a ROS container, aggregate PARTITIONS over the ros_id column.

Example

See Viewing Partition Storage Data in the Administrator's Guide.

**PROCESS SIGNALS**

Returns a history of signals that were received and handled by the Vertica process. For details about signals, see the Linux documentation.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica recorded the signal.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>SIGNAL_NUMBER</td>
<td>INTEGER</td>
<td>Signal number, refers to POSIX SIGNAL_NUMBER</td>
</tr>
<tr>
<td>SIGNAL_CODE</td>
<td>INTEGER</td>
<td>Signal code.</td>
</tr>
<tr>
<td>SIGNAL_PID</td>
<td>INTEGER</td>
<td>Linux process identifier of the signal.</td>
</tr>
<tr>
<td>SIGNAL_UID</td>
<td>INTEGER</td>
<td>Process ID of sending process.</td>
</tr>
<tr>
<td>SIGNAL_ADDRESS</td>
<td>INTEGER</td>
<td>Address at which fault occurred.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser
PROJECTION_RECOVERIES

Retains history about projection recoveries. Because Vertica adds an entry per recovery plan, a projection/node pair might appear multiple times in the output.

**Note:** You cannot query this or other system tables during cluster recovery; the cluster must be UP to accept connections.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is recovering or has recovered the corresponding projection.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Name of the projection that is being or has been recovered on the corresponding node.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any. TRANSACTION_ID initializes as NO_TRANSACTION with a value of 0. Vertica will ignore the recovery query and keep (0) if there's no action to take (no data in the table, etc). When no recovery transaction starts, ignored value appears in this table's STATUS column.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>METHOD</td>
<td>VARCHAR</td>
<td>Recovery method that Vertica chooses. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• incremental</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• incremental-replay-delete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• split</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recovery-by-container</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| STATUS      | VARCHAR   | Current projection-recovery status on the corresponding node. STATUS can be "queued," which indicates a brief period between the time the query is prepared and when it runs. Possible values are:  
  - queued  
  - running  
  - finished  
  - ignored  
  - error-retry  
  - error-fatal |
| PROGRESS    | INTEGER   | An estimate (value in the range [0,100]) of percent complete for the recovery task described by this information.  
  Note: The actual amount of time it takes to complete a recovery task depends on a number of factors, including concurrent workloads and characteristics of the data; therefore, accuracy of this estimate can vary.  
  The PROGRESS column value is NULL after the task completes. |
| DETAIL      | VARCHAR   | More detailed information about PROGRESS. The values returned for this column depend on the type of recovery plan:  
  - General recovery plans – value displays the estimated progress, as a percent, of the three primary parts of the plan: Scan, Sort, and Write.  
  - Recovery-by-container plans – value begins with CopyStorage: and is followed by the number of bytes copied over the total number of bytes to copy. |
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replay</td>
<td></td>
<td>• Replay delete plans – value begins with Delete: and is followed by the number of deletes replayed over an estimate of the total number of deletes to replay. The DETAIL column value becomes NULL after the recovery plan completes.</td>
</tr>
<tr>
<td>start_time</td>
<td>TIMESTAMPTZ</td>
<td>Time the recovery task described by this information started.</td>
</tr>
<tr>
<td>end_time</td>
<td>TIMESTAMPTZ</td>
<td>Time the recovery task described by this information ended.</td>
</tr>
<tr>
<td>runtime_priority</td>
<td>VARCHAR</td>
<td>Determines the amount of runtime resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HIGH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MEDIUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOW</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**See Also**

*RECOVERY_STATUS*

**PROJECTION_REFRESHES**

System table PROJECTION_REFRESHES records information about refresh operations, successful and unsuccessful. PROJECTION_REFRESHES retains refresh data until one of the
following events occurs:

- **CLEAR_PROJECTION_REFRESHES** is called.
- The table's storage quota is exceeded.

Tables and projections can be dropped while a query runs against them. The query continues to run, even after the drop occurs. Only when the query finishes does it notice the drop, which might cause a rollback. The same is true for refresh queries. Thus, **PROJECTION_REFRESHES** might report that a projection failed to be refreshed before the refresh query completes. In this case, the **REFRESH_DURATION_SEC** column continues to increase until the refresh query completes.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node where the refresh was initiated.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the projection schema.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Name of the refreshed projection.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the projection's anchor table.</td>
</tr>
<tr>
<td>REFRESH_STATUS</td>
<td>VARCHAR</td>
<td>Status of refresh operations for this projection, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Queued : Projection is queued for refresh.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Refreshing: Projection refresh is in progress.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Refreshed: Projection refresh is complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Failed: Projection refresh failed.</td>
</tr>
<tr>
<td>REFRESH_PHASE</td>
<td>VARCHAR</td>
<td>Indicates how far the refresh has progressed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Historical: Refresh reached the first phase and is refreshing data from historical data. This refresh phase requires the most amount of time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Current: Refresh reached the final phase and is</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attempting to refresh data from the current epoch. To complete this phase, refresh must obtain a lock on the table. If the table is locked by another transaction, refresh is blocked until that transaction completes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The LOCKS system table is useful for determining if a refresh is blocked on a table lock. To determine if a refresh has been blocked, locate the term &quot;refresh&quot; in the transaction description. A refresh has been blocked when the scope for the refresh is REQUESTED and other transactions acquired a lock on the table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This field is NULL until the projection starts to refresh and is NULL after the refresh completes.</td>
</tr>
<tr>
<td>REFRESH_METHOD</td>
<td>VARCHAR</td>
<td>Method used to refresh the projection:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Buddy: Projection refreshed from the contents of a buddy projection. This method maintains historical data, so the projection can used for historical queries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Scratch: Projection refreshed without using a buddy projection. This method does not generate historical data, so the projection cannot participate in historical queries on data that precedes the refresh.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rebalance: If the projection is segmented, it is refreshed from scratch; if unsegmented, it is refreshed from a buddy projection.</td>
</tr>
<tr>
<td>REFRESH_FAILURE_COUNT</td>
<td>INTEGER</td>
<td>Number of times a refresh failed for the projection. REFRESH_FAILURE_COUNT does not indicate whether the projection was eventually refreshed. See REFRESH_STATUS to determine whether the refresh operation is progressing.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the refresh session.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REFRESH_START</td>
<td>TIMESTAMPTZ</td>
<td>Time the projection refresh started.</td>
</tr>
<tr>
<td>REFRESH_DURATION_SEC</td>
<td>INTERVAL SECOND (0)</td>
<td>How many seconds the projection refresh ran.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Differentiates active and completed refresh operations.</td>
</tr>
</tbody>
</table>
| RUNTIME_PRIORITY         | VARCHAR         | Determines how many run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool, one of the following:  
  - HIGH  
  - MEDIUM  
  - LOW |
| TRANSACTION_ID           | INTEGER         | Identifier for the transaction within the session, if any; otherwise NULL. |

**Note:** The transaction_id is correlated with the execution plan only when refreshing from scratch. When refreshing from a buddy, multiple sub-transactions are created.

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**PROJECTION_STORAGE**

Monitors the amount of disk storage used by each projection on each node.

**Note:** Projections that have no data never have full statistics. Querying this system table lets you see if your projection contains data.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION SCHEMA</td>
<td>VARCHAR</td>
<td>The name of the schema associated with the projection.</td>
</tr>
<tr>
<td>PROJECTION_COLUMN_COUNT</td>
<td>INTEGER</td>
<td>The number of columns in the projection.</td>
</tr>
<tr>
<td>ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in the table's projections, including any rows marked for deletion.</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes of disk storage used by the projection.</td>
</tr>
<tr>
<td>WOS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of WOS rows in the projection.</td>
</tr>
<tr>
<td>WOS_USED_BYTES</td>
<td>INTEGER</td>
<td>The number of WOS bytes in the projection.</td>
</tr>
<tr>
<td>ROS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of ROS rows in the projection.</td>
</tr>
<tr>
<td>ROS_USED_BYTES</td>
<td>INTEGER</td>
<td>The number of ROS bytes in the projection.</td>
</tr>
<tr>
<td>ROS_COUNT</td>
<td>INTEGER</td>
<td>The number of ROS containers in the projection.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>The associated table name for which information is listed.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>The associated table schema for which information is listed.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID, assigned by the Vertica catalog, which identifies the anchor table.</td>
</tr>
</tbody>
</table>

See Also

- PROJECTIONS
- ANALYZE_STATISTICS
**PROJECTION_USAGE**

Records information about projections Vertica used in each processed query.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_START_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Value of query at beginning of history interval.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>REQUEST_ID</td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>IO_TYPE</td>
<td>VARCHAR</td>
<td>Input/output.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Projection name for which information is listed.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica, which identifies the anchor table.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Name of the schema that contains the anchor table.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the projection's associated anchor table.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**QUERY_EVENTS**

Returns information about query planning, optimization, and execution events.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica recorded the event.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Identifier of the user for the query event.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user for which Vertica lists query information at the time it recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>REQUEST_ID</td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EVENT_CATEGORY</td>
<td>VARCHAR</td>
<td>Category of event: OPTIMIZATION or EXECUTION.</td>
</tr>
<tr>
<td>EVENT_TYPE</td>
<td>VARCHAR</td>
<td>Type of event. For more information, refer to the Event Types table on this page.</td>
</tr>
<tr>
<td>EVENT_DESCRIPTION</td>
<td>VARCHAR</td>
<td>Generic description of the event.</td>
</tr>
<tr>
<td>OPERATOR_NAME</td>
<td>VARCHAR</td>
<td>Name of the Execution Engine component that generated the event, if applicable; for example, NetworkSend. Values from the OPERATOR_NAME and PATH_ID columns let you tie a query event back to a particular operator in the query plan. If the event did not come from a specific operator, the OPERATOR_NAME column is NULL.</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>INTEGER</td>
<td>Unique identifier that Vertica assigns to a query operation or path in a query plan. If the event did not come from a specific operator, the PATH_ID column is NULL. See EXECUTION ENGINE PROFILES for more information.</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>INTEGER</td>
<td>Object identifier (such as projection or table) to which the event refers.</td>
</tr>
<tr>
<td>EVENT_DETAILS</td>
<td>VARCHAR</td>
<td>Free-form text describing the specific event.</td>
</tr>
<tr>
<td>SUGGESTED_ACTION</td>
<td>VARCHAR</td>
<td>Suggested user action, if any is available.</td>
</tr>
</tbody>
</table>

The following table lists informational event types.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP_FALLBACK</td>
<td>This optimization did not apply to this query type.</td>
</tr>
<tr>
<td>SMALL_MERGE_REPLACED</td>
<td>Vertica has chosen a more efficient way to access the data by replacing a merge.</td>
</tr>
<tr>
<td>STORAGE_CONTAINERS_ELIMINATED</td>
<td>Vertica has performed partition pruning for the purpose of optimization.</td>
</tr>
<tr>
<td>GROUPBY_PUSHDOWN</td>
<td>(Message is Internal to Vertica.)</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NO GROUPBY PUSHDOWN</td>
<td>(Message is Internal to Vertica.)</td>
</tr>
<tr>
<td>VALUE_TRUNCATED</td>
<td>A character value is too long.</td>
</tr>
<tr>
<td>GROUP_BY_PREPASS_FALLBACK</td>
<td>Vertica could not run an optimization. In-memory prepass is disabled. The projection may not be optimal.</td>
</tr>
<tr>
<td>MERGE_CONVERTED_TO_UNION</td>
<td>Vertica has converted a merge operator to a union operator due to the sort order of the multi-threaded storage access stream.</td>
</tr>
<tr>
<td>TRANSITIVE PREDICATE</td>
<td>Vertica has optimized by adding predicates to joins where it makes logical sense to do so.</td>
</tr>
<tr>
<td></td>
<td>For example, for the statement, SELECT * FROM A, B WHERE A.a = B.a AND A.a = 1; Vertica may add a predicate B a = 1 as a filter for better storage access of table B.</td>
</tr>
<tr>
<td>NODE PRUNING</td>
<td>Vertica has performed node pruning, which is similar to partition pruning, but at the node level.</td>
</tr>
<tr>
<td>SEQUENCE CACHE REFILLED</td>
<td>Vertica has refilled sequence cache.</td>
</tr>
<tr>
<td>OUTER OVERRIDE USED</td>
<td>For efficiency and optimization, Vertica has swapped the inner/outer tables in a join. Vertica used the smaller table as the inner table.</td>
</tr>
<tr>
<td>OUTER OVERRIDE NOT USED</td>
<td>Vertica found swapping inner/outer tables in a join unnecessary because the inner/outer tables were in good order. (For example, a smaller table was used in an inner join.)</td>
</tr>
<tr>
<td>EXTERNAL_PREDICATE_PUSHDOWN_NOT_SUPPORTED</td>
<td>Predicate pushdown for older Hive versions may not be supported. For more information, see Improving Query Performance for Data Stored in HDFS.</td>
</tr>
<tr>
<td>LibHDFS++ UNSUPPORTED OPERATION</td>
<td>Vertica accessed HDFS using the hdfs URL scheme, but the HDFS cluster uses an unsupported feature such as wire encryption or HTTPS_ONLY. Vertica fell back to WebHDFS.</td>
</tr>
</tbody>
</table>
Vertica attempted to contact a Name Node on an HDFS cluster that uses High Availability Name Node and did not receive a response. Vertica retried with a different Name Node.

The following table lists event types that you should review for possible corrective action.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_BY_SPIPPED</td>
<td>This event type is typically related to a specific type of query, which you may need to adjust.</td>
<td>Identify the type of query and make adjustments accordingly. You may need to adjust resource pools, projections, or the amount of RAM available. Try running the query on a cluster with no additional workload.</td>
</tr>
<tr>
<td>RESEGMENTED_MANY_ROWS</td>
<td>This event type is typically related to a specific type of query, which you may need to adjust.</td>
<td>Do projections need to be segmented in a different way to allow for join locality? Can you rewrite the query to filter out more rows at storage access time? (Typically, Vertica does so automatically through predicate pushdown.) Review your explain plan.</td>
</tr>
<tr>
<td>RLE_OVERRIDDEN</td>
<td>The average run counts are not large enough for Run Length Encoding (RLE). This event occurs with queries where the filtered results for certain columns do not work with RLE because cardinality is less than 10.</td>
<td>Review and rewrite your query, if necessary.</td>
</tr>
<tr>
<td>PREDICATE OUTSIDE HISTOGRAM</td>
<td>A predicate value you are trying to match does not exist in a set</td>
<td>Review and rewrite your query, if necessary.</td>
</tr>
</tbody>
</table>
of possible values for a specific column. For example, you try to match a VARCHAR value WHERE mystring = "ABC\newline". In this case, the newline character throws off the predicate matching optimizations.

The following table lists more critical event types, and recommends actions you can take to resolve issues.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WOS_SPILL</td>
<td>WOS ran out of memory, and began spilling to ROS.</td>
<td>Try one or more of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adjust the load process to run bigger batches less frequently.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Give the WOS resource pool more memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Increase how often the Tuple Mover moveout and mergeout processes run for high ingest rates. (Adjustment of the moveout process helps avoid ROS issues.)</td>
</tr>
<tr>
<td>NO HISTOGRAM</td>
<td>Indicates a table does not have an updated column histogram.</td>
<td>Running the function ANALYZE_STATISTICS most often corrects this issue.</td>
</tr>
<tr>
<td>MEMORY LIMIT HIT</td>
<td>Indicates query complexity or, possibly, lack of available system memory.</td>
<td>Consider adjusting the MAXMEMORYSIZE and PLANNEDCONCURRENCY resource pools so that the optimizer has sufficient memory. On a heavily used system, this event may occur more frequently.</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
<td>Action</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DELETE WITH NON OPTIMIZED PROJECTION</td>
<td>One or more projections do not have your delete filter column in their sort order, causing Vertica difficulty identifying rows to mark as deleted.</td>
<td>Add the delete filter column to the end of every projection sort order for your target delete table.</td>
</tr>
<tr>
<td>JOIN_SPILLED</td>
<td>Vertica has spilled a join to disk. A join spill event slows down the subject query and all other queries as it consumes resources while using disk as virtual memory.</td>
<td>Try the following (in sequence):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Review the explain plan. The query could be too ambitious, for example, cross joining two large tables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Consider adding the query to a lower priority pool to reduce impact on other queries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Create projections that allow for a merge join instead of a hash join.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adjust the PLANNEDCONCURRENCY resource pool so that queries have more memory to execute.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.
See Also

- `EXECUTION_ENGINE_PROFILES`
- `QUERY_PLAN_PROFILES`

**QUERY_METRICS**

Monitors the sessions and queries running on each node.

**Note**: Totals in this table are reset each time the database restarts.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>ACTIVE_USER_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The number of active user sessions (connections).</td>
</tr>
<tr>
<td>ACTIVE_SYSTEM_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The number of active system sessions.</td>
</tr>
<tr>
<td>TOTAL_USER_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The total number of user sessions.</td>
</tr>
<tr>
<td>TOTAL_SYSTEM_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The total number of system sessions.</td>
</tr>
<tr>
<td>TOTAL_ACTIVE_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The total number of active user and system sessions.</td>
</tr>
<tr>
<td>TOTAL_SESSION_COUNT</td>
<td>INTEGER</td>
<td>The total number of user and system sessions.</td>
</tr>
<tr>
<td>RUNNING_QUERY_COUNT</td>
<td>INTEGER</td>
<td>The number of queries currently running.</td>
</tr>
<tr>
<td>EXECUTED_QUERY_COUNT</td>
<td>INTEGER</td>
<td>The total number of queries that ran.</td>
</tr>
</tbody>
</table>

**QUERY_PLAN_PROFILES**

Provides detailed execution status for queries that are currently running in the system. Output from the table shows the real-time flow of data and the time and resources consumed for each path in each query plan.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>An identifier for the transaction within the session if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session; these columns are useful for creating joins with other system tables.</td>
</tr>
<tr>
<td>PATH_ID</td>
<td>INTEGER</td>
<td>Unique identifier that Vertica assigns to a query operation or path in a query plan. Textual representation for this path is output in the PATH_LINE column.</td>
</tr>
<tr>
<td>PATH_LINE_INDEX</td>
<td>INTEGER</td>
<td>Each plan path in QUERY_PLAN_PROFILES could be represented with multiple rows. PATH_LINE_INDEX returns the relative line order. You should include the PATH_LINE_INDEX column in the QUERY_PLAN_PROFILES ... ORDER BY clause so rows in the result set appear as they do in EXPLAIN-generated query plans.</td>
</tr>
<tr>
<td>PATH_IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Status of a path in the query plan. True (t) if the path has started running, otherwise false.</td>
</tr>
<tr>
<td>PATH_IS_COMPLETE</td>
<td>BOOLEAN</td>
<td>Status of a path in the query plan. True (t) if the path has finished running, otherwise false.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Status of a running query. True if the query is currently active (t), otherwise false (f).</td>
</tr>
<tr>
<td>RUNNING_TIME</td>
<td>INTERVAL</td>
<td>The amount of elapsed time the query path took to execute.</td>
</tr>
<tr>
<td>MEMORY_ALLOCATED_BYTES</td>
<td>INTEGER</td>
<td>The amount of memory the path used, in bytes.</td>
</tr>
<tr>
<td>READ_FROM_DISK_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes the path read from disk (or the disk cache).</td>
</tr>
<tr>
<td>RECEIVED_BYTES</td>
<td>INTEGER</td>
<td>The number of bytes received over the network.</td>
</tr>
<tr>
<td>SENT_BYTES</td>
<td>INTEGER</td>
<td>Size of data sent over the network by the path.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PATH_LINE</td>
<td>VARCHAR</td>
<td>The query plan text string for the path, associated with the PATH ID and PATH_LINE_INDEX columns.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit permissions are required; however, users see only the records that correspond to tables they have permissions to view.

**Best Practices**

Table results can be very wide. For best results when you query the QUERY_PLAN_PROFILES table, sort on these columns:

- TRANSACTION_ID
- STATEMENT_ID
- PATH_ID
- PATH_LINE_INDEX

For example:

```
=> SELECT ... FROM query_plan_profiles
   WHERE ...
   ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

**Example**

See [Profiling Query Plans](#) in the Administrator's Guide

**See Also**

- [EXECUTION_ENGINE_PROFILES](#)
- [EXPLAIN](#)
### QUERY_PROFILES

Provides information about executed queries.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>The identification of the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>An identifier for the transaction within the session if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>IDENTIFIER</td>
<td>VARCHAR</td>
<td>A string to identify the query in system tables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> You can query the IDENTIFIER column to quickly identify queries you have labeled for profiling and debugging. See Labeling Queries in the Administrator’s Guide for details.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>QUERY</td>
<td>VARCHAR</td>
<td>The query string used for the query.</td>
</tr>
<tr>
<td>QUERY_SEARCH_PATH</td>
<td>VARCHAR</td>
<td>A list of schemas in which to look for tables.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name in which the query is being profiled, set only for load operations.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The table name in the query being profiled, set only for load operations.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>QUERY_DURATION_US</td>
<td>NUMERIC (18, 0)</td>
<td>The duration of the query in microseconds.</td>
</tr>
<tr>
<td>QUERY_START_EPOCH</td>
<td>INTEGER</td>
<td>The epoch number at the start of the given query.</td>
</tr>
<tr>
<td>QUERY_START</td>
<td>VARCHAR</td>
<td>The Linux system time of query execution in a format that can be used as a DATE/TIME expression.</td>
</tr>
<tr>
<td>QUERY_TYPE</td>
<td>VARCHAR</td>
<td>Is one of INSERT, SELECT, UPDATE, DELETE, UTILITY, or UNKNOWN.</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td>INTEGER</td>
<td>The return error code for the query.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user who ran the query.</td>
</tr>
<tr>
<td>PROCESSED_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows returned by the query.</td>
</tr>
<tr>
<td>RESERVED_EXTRA_MEMORY_B</td>
<td>INTEGER</td>
<td>Shows how much unused memory (in bytes) remains that is reserved for a given query but is unassigned to a specific operator. This is the memory from which unbounded operators pull first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Displays information about actively running queries, regardless of whether profiling is enabled.</td>
</tr>
</tbody>
</table>

**QUERY_REQUESTS**

Returns information about user-issued query requests.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user who issued the query at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>REQUEST_ID</td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>REQUEST_TYPE</td>
<td>VARCHAR</td>
<td>Type of the query request. Examples include, but are not limited to: QUERY, DDL, LOAD, UTILITY, TRANSACTION, PREPARE, EXECUTE, SET, SHOW</td>
</tr>
<tr>
<td>REQUEST</td>
<td>VARCHAR</td>
<td>Query statement.</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
REQUEST_LABEL | VARCHAR | Label of the query, if available.
SEARCH_PATH | VARCHAR | Contents of the search path.
MEMORY_ACQUIRED_MB | FLOAT | Memory acquired by this query request in megabytes.
SUCCESS | BOOLEAN | Value returned if the query successfully executed.
ERROR_COUNT | INTEGER | Number of errors encountered in this query request (logged in ERROR_MESSAGES table).
START_TIMESTAMP | TIMESTAMPTZ | Beginning of history interval.
END_TIMESTAMP | TIMESTAMPTZ | End of history interval.
REQUEST_DURATION | TIMESTAMPTZ | Length of time the query ran in days, hours, minutes, second, and milliseconds.
REQUEST_DURATION_MS | INTEGER | Length of time the query ran in milliseconds.
IS_EXECUTING | BOOLEAN | Distinguishes between actively-running (t) and completed (f) queries.

### Privileges
No explicit privileges are required. You can only see the records for requests associated with your user name. Users with SYSMONITOR Role or DBADMIN Role can see all records from all users.

### Example

```sql
=> SELECT * FROM QUERY_REQUESTS;
```

```
node_name | v_vmart_node0001
user_name | dbadmin
session_id | v_vmart_node0001-321039:0x3a
request_id | 404
transaction_id | 45035996274380645
statement_id | 1
request_type | DDL
request | GRANT EXECUTE ON FUNCTION public.STV_SetExportShapefileDirectory() TO dbadmin;
request_label |
search_path | "$user", public, v_catalog, v_monitor, v_internal
```
See Also
QUERY_PROFILES

REBALANCE_OPERATIONS

Contains information on historic and ongoing rebalance operations.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| OBJECT_TYPE     | VARCHAR   | The type of the object being rebalanced. Valid types are:  
|                 |           | • Projection  
<p>|                 |           | • DFSfile     |
| OBJECT_ID       | INTEGER   | The ID of the object being rebalanced. |
| OBJECT_NAME     | VARCHAR   | The name of the object being rebalanced. Objects can be tables, projections, or other Vertica objects. |
| PATH_NAME       | VARCHAR   | The DFS path for unstructured data being rebalanced. |
| TABLE_NAME      | VARCHAR   | The name of the table being rebalanced. This value is NULL for DFS files. |
| TABLE_SCHEMA    | VARCHAR   | The schema of the table being rebalanced. This value is NULL for DFS files. |
| TRANSACTION_ID  | INTEGER   | The identifier for the transaction within the session. |
| STATEMENT_ID    | INTEGER   | The unique numeric ID for the currently- |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>running statement.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is rebalancing.</td>
</tr>
<tr>
<td>OPERATION_NAME</td>
<td>VARCHAR</td>
<td>Identifies the specific rebalance operation being performed, set to one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Refresh projection, update <em>temporary projection name</em> name and ID to <em>master projection name</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Drop unsegmented replicas</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Replicate DFS File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Refresh projection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Drop replaced or replacement projection, rename <em>temporary projection name</em> to <em>original projection name</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update temp table segments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Prepare: moveout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Prepare: separate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Move storage containers</td>
</tr>
<tr>
<td>OPERATION_STATUS</td>
<td>VARCHAR</td>
<td>Returns Running or an empty string to indicate 'not running.' One of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- START</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COMPLETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ABORT</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>When TRUE, the operation is currently running.</td>
</tr>
<tr>
<td>REBALANCE_METHOD</td>
<td>VARCHAR</td>
<td>The method that Vertica is using to perform the rebalance. Valid methods are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- REFRESH: New projections are created</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>according to the new segmentation definition. Data is copied via a refresh plan from projections with the previous segmentation to the new segments. This method is only used if specifically requested (using START_REFRESH), by setting a configuration parameter, or if Elastic Cluster is disabled or if there is a change in desired k-safety.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REPLICATE: Unsegmented projection data is copied to new nodes and removed from ephemeral nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ELASTIC_CLUSTER: The segmentation of existing segmented projections is altered to adjust to a new cluster topology and data is redistributed accordingly.</td>
</tr>
</tbody>
</table>

- **SESSION_ID**   VARCHAR   Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.  
- **OPERATION_START_TIMESTAMP**   TIMESTAMPZ   The time that the rebalance began.  
- **OPERATION_END_TIMESTAMP**   TIMESTAMPZ   The time that the rebalance ended. If the rebalance is ongoing, this value is NULL.  
- **ELASTIC_CLUSTER_VERSION**   INTEGER   The Elastic Cluster has a version. Each time the cluster topology changes, this version increments.  
- **IS_LATEST**   BOOLEAN   True if this row pertains to the most recent rebalance activity.

**Privileges**

Superuser
**REBALANCE_PROJECTION_STATUS**

Maintain history on rebalance progress for relevant projections.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Identifier of the projection that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>PROJECTION_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema of the projection that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Name of the projection that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_ID</td>
<td>INTEGER</td>
<td>Anchor table identifier of the projection that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>ANCHOR_TABLE_NAME</td>
<td>VARCHAR</td>
<td>Anchor table name of the projection that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>REBALANCE_METHOD</td>
<td>VARCHAR</td>
<td>Method that was, is, or will be used to rebalance the projection. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- REFRESH: New projections are created according to the new segmentation definition. Data is copied via a refresh plan from projections with the previous segmentation to the new segments. This method is only used if specifically requested (using <code>START_REFRESH</code>), by setting a configuration parameter, or if Elastic Cluster is disabled or if there is a change in desired k-safety.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- REPLICATE: Unsegmented projection data is copied to new nodes and removed from ephemeral nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ELASTIC_CLUSTER: The segmentation of existing segmented projections is altered to adjust to a new cluster topology and data is redistributed accordingly.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DURATION_SEC</td>
<td>INTERVAL_SEC</td>
<td>Deprecated - populated by NULL. Length of time (seconds) rebalance has been working on this projection, including time to separate storage, if that work is required.</td>
</tr>
<tr>
<td>SEPARATED_PERCENT</td>
<td>NUMERIC (5,2)</td>
<td>Percent of storage that has been separated for this projection.</td>
</tr>
<tr>
<td>TRANSFERRED_PERCENT</td>
<td>NUMERIC (5,2)</td>
<td>Percent of storage that has been transferred, for this projection.</td>
</tr>
<tr>
<td>SEPARATED_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes, separated by the corresponding rebalance operation, for this projection.</td>
</tr>
<tr>
<td>TO_SEPARATE_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes that remain to be separated by the corresponding rebalance operation for this projection.</td>
</tr>
<tr>
<td>TRANSFERRED_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes transferred by the corresponding rebalance operation for this projection.</td>
</tr>
<tr>
<td>TO_TRANSFER_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes that remain to be transferred by the corresponding rebalance operation for this projection.</td>
</tr>
<tr>
<td>IS_LATEST</td>
<td>BOOLEAN</td>
<td>True if this row pertains to the most recent rebalance activity, where elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster);</td>
</tr>
<tr>
<td>ELASTIC_CLUSTER_VERSION</td>
<td>INTEGER</td>
<td>The Elastic Cluster has a version, and each time the cluster topology changes, this version is incremented. This column reflects the version to which this row of information pertains. The TO_* fields (TO_SEPARATE_* and TO_TRANSFER_*) are only valid for the current version. To view only rows from the current, latest or upcoming rebalance operation, use: WHERE elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster);</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>INTEGER</td>
<td>Identifier of the table that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema of the table that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Name of the table that will be, was, or is being rebalanced.</td>
</tr>
<tr>
<td>REBALANCE_METHOD</td>
<td>VARCHAR</td>
<td>Method that will be, is, or was used to rebalance the projections of this table. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- REFRESH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- REPLICATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ELASTIC_CLUSTER</td>
</tr>
<tr>
<td>DURATION_SEC</td>
<td>INTERVAL SEC</td>
<td>Deprecated - populated by NULL. Aggregate, by table_id, rebalance_method, and</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>elastic_cluster_version</td>
<td></td>
<td>elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>SEPARATED_PERCENT</td>
<td>NUMERIC(5,2)</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>TRANSFERRED_PERCENT</td>
<td>NUMERIC(5,2)</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>SEPARATED_BYTES</td>
<td>INTEGER</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>TO_SEPARATE_BYTES</td>
<td>INTEGER</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>TRANSFERRED_BYTES</td>
<td>INTEGER</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>TO_TRANSFER_BYTES</td>
<td>INTEGER</td>
<td>Aggregate, by table_id, rebalance_method, and elastic_cluster_version, of the same in REBALANCE_PROJECTION_STATUS.</td>
</tr>
<tr>
<td>IS_LATEST</td>
<td>BOOLEAN</td>
<td>True if this row pertains to the most recent rebalance activity, where elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster;)</td>
</tr>
<tr>
<td>ELASTIC_CLUSTER_VERSION</td>
<td>INTEGER</td>
<td>The Elastic Cluster has a version, and each time the cluster topology changes, this version is incremented. This column reflects the version to which this row of information pertains. The TO_* fields (TO_SEPARATE_* and TO_TRANSFER_*) are only valid for the current version. To view only rows from the current, latest or upcoming rebalance operation, use:</td>
</tr>
</tbody>
</table>
**WHERE** elastic_cluster_version = (SELECT version FROM v_catalog.elastic_cluster;)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_timestamp</td>
<td>TIMESTAMPZ</td>
<td>The time that the rebalance began.</td>
</tr>
<tr>
<td>end_timestamp</td>
<td>TIMESTAMPZ</td>
<td>The time that the rebalance ended.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser

**See Also**

- ELASTIC_CLUSTER
- REBALANCE_PROJECTION_STATUS

**RECOVERY_STATUS**

Provides the status of recovery operations, returning one row for each node.

**Note:** You cannot query this or other system tables table during cluster recovery; the cluster must be UP to accept connections.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>RECOVER_EPOCH</td>
<td>INTEGER</td>
<td>Epoch the recovery operation is trying to catch up to.</td>
</tr>
<tr>
<td>RECOVERY_PHASE</td>
<td>VARCHAR</td>
<td>Current stage in the recovery process. Can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• current</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
---|---|---
| | | • historical pass $X$, where $X$ is the iteration count
| SPLITS_COMPLETED | INTEGER | Number of independent recovery SPLITs queries that have run and need to run.
| SPLITs_TOTAL | INTEGER | Total number of SPLITs queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If SPLITs_TOTAL = 2, then there should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
| HISTORICAL_COMPLETED | INTEGER | Number of independent recovery HISTORICAL queries that have run and need to run.
| HISTORICAL_TOTAL | INTEGER | Total number of HISTORICAL queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If HISTORICAL_TOTAL = 2, then there should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
| CURRENT_COMPLETED | INTEGER | Number of independent recovery CURRENT queries that have run and need to run.
| CURRENT_TOTAL | INTEGER | Total number of CURRENT queries that ran. Each query corresponds to one row in the PROJECTION_RECOVERIES table. If CURRENT_TOTAL = 2, then there should be 2 rows added to PROJECTION_RECOVERIES, showing query details.
| IS_RUNNING | BOOLEAN | True (t) if the node is still running recovery; otherwise false (f).

### Privileges

None

### See Also

PROJECTION_RECOVERIES
REMOTE_REPLICATION_STATUS

Provides the status of replication tasks to alternate clusters.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_EPOCH</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>EPOCH</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LAST_REPLICATED_TIME</td>
<td>TIMESTAMPZ</td>
<td></td>
</tr>
<tr>
<td>OBJECTS</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>REPLICATED_EPOCH</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>REPLICATION_POINT</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>SNAPSHOT_NAME</td>
<td>VARCHAR</td>
<td></td>
</tr>
</tbody>
</table>

Privileges

None

REPARENTED_ON_DROP

Lists re-parenting events of objects dropped from their original owner but still remain in Vertica. For example, a user may leave the organization and needs to be removed from the database. When the dbadmin drops the user from the database that user's objects are re-parented to another user.

In some cases a Vertica user's objects are reassigned based on the GlobalHeirUserName parameter. In this case a user's objects are re-parented to the user indicated by this parameter.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPARENT_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The time the re-parenting event occurred.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node or nodes on which the re-parenting occurred.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>The identification number of the re-parenting event.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>The unique, system-generated user identification number.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user that caused the re-parenting event. For example, a dbadmin user may have dropped a user thus re-parenting that user’s objects.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>The system-generated transaction identification number. Is NULL if a transaction id does not exist.</td>
</tr>
<tr>
<td>OLD_OWNER_NAME</td>
<td>VARCHAR</td>
<td>The name of the dropped user who used to own the re-parented object.</td>
</tr>
<tr>
<td>OLD_OWNER_OID</td>
<td>INTEGER</td>
<td>The unique identification number of the user who used to own the re-parented object.</td>
</tr>
<tr>
<td>NEW_OWNER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user who now owns the re-parented objects.</td>
</tr>
<tr>
<td>NEW_OWNER_OID</td>
<td>INTEGER</td>
<td>The unique identification number of the user who now owns the re-parented objects.</td>
</tr>
<tr>
<td>OBJ_NAME</td>
<td>VARCHAR</td>
<td>The name of the object being re-parented.</td>
</tr>
<tr>
<td>OBJ_OID</td>
<td>INTEGER</td>
<td>The unique identification number of the object being re-parented.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema in which the object resides.</td>
</tr>
<tr>
<td>SCHEMA_OID</td>
<td>INTEGER</td>
<td>The unique identification number of the schema in which the re-parented object resides.</td>
</tr>
</tbody>
</table>

**RESOURCE_ACQUISITIONS**

Retains information about resources (memory, open file handles, threads) acquired by each running request for each resource pool in the system.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node name for which information is listed.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Transaction identifier for this request.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>REQUEST_TYPE</td>
<td>VARCHAR</td>
<td>Type of request issued to a resource pool. Request type can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Reserve: related to queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Acquire: [Internal] related to the optimizer and other internal services, such as the Database Designer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Acquire additional: [Internal] related to size adjustment of acquisitions obtained through the first two methods; unusual, outside the WOS</td>
</tr>
<tr>
<td>POOL_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the resource pool. The ID represents the initial pool, even if the request has been moved to a new pool due to cascade. For more information about cascade events, see RESOURCE_POOL_MOVE.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>Name of the resource pool. The name represents the initial pool, even if the request has been moved to a new pool due to cascade. For more information about cascade events, see RESOURCE_POOL_MOVE.</td>
</tr>
<tr>
<td>THREAD_COUNT</td>
<td>INTEGER</td>
<td>Number of threads in use by this request.</td>
</tr>
<tr>
<td>OPEN_FILE_HANDLE_COUNT</td>
<td>INTEGER</td>
<td>Number of open file handles in use by this request.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MEMORY_INUSE_KB</td>
<td>INTEGER</td>
<td>Total amount of memory in kilobytes acquired by this query. Column RESERVED_EXTRA_MEMORY_B in system table QUERY_PROFILES shows how much unused memory (in bytes) remains that is reserved for a given query but is unassigned to a specific operator. If operators for a query acquire all memory specified by MEMORY_INUSE_KB, the plan must request more memory from the Vertica resource manager.</td>
</tr>
<tr>
<td>QUEUE_ENTRY_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Timestamp when the request was queued at the Resource Manager.</td>
</tr>
<tr>
<td>ACQUISITION_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Timestamp when the request was admitted to run. See the Notes section below for the difference between these two timestamps.</td>
</tr>
<tr>
<td>RELEASE_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica released this resource acquisition.</td>
</tr>
<tr>
<td>DURATION_MS</td>
<td>INTEGER</td>
<td>Duration of the resource request in milliseconds.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Denotes if the query holding the resource is still executing (t).</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**Notes**

When monitoring resource pools and resource usage by queries, the “queue wait” time is the difference between ACQUISITION_TIMESTAMP and QUEUE_ENTRY_TIMESTAMP. For example, to determine how long a query waits in the queue before it is admitted to run, you
can get the difference between the ACQUISITION_TIMESTAMP and the QUEUE_ENTRY_TIMESTAMP using a query like the following:

```sql
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp, 
   (acquisition_timestamp - queue_entry_timestamp) AS 'queue wait'
FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE 'node0001';
```

See Also

- RESOURCE_POOL_STATUS
- RESOURCE_POOLS
- RESOURCE_QUEUES
- RESOURCE_REJECTIONS

**RESOURCE_POOL_MOVE**

Displays the cascade event information on each node.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node name for which information is listed.</td>
</tr>
<tr>
<td>MOVE_TIMESTAMP</td>
<td>TIMESTAMPZ</td>
<td>Time when the query attempted to move to the target pool.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Identifies the query event user.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user for which Vertica lists query information at the time it records the session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Transaction identifier for the request.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the statement.</td>
</tr>
<tr>
<td>SOURCE_POOL_NAME</td>
<td>VARCHAR</td>
<td>Name of the resource pool where the query was executing when Vertica attempted the move.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TARGET_POOL_NAME</td>
<td>VARCHAR</td>
<td>Name of resource pool where the query attempted to move.</td>
</tr>
<tr>
<td>MOVE_CAUSE</td>
<td>VARCHAR</td>
<td>Denotes why the query attempted to move.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MOVE RESOURCE POOL COMMAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RUNTIMECAP EXCEEDED</td>
</tr>
<tr>
<td>SOURCE_CAP</td>
<td>INTEGER</td>
<td>Effective RUNTIMECAP value for the source pool. The value represents the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lowest of these three values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- session RUNTIMECAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- user RUNTIMECAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- source pool RUNTIMECAP</td>
</tr>
<tr>
<td>TARGET_CAP</td>
<td>INTEGER</td>
<td>Effective RUNTIMECAP value for the target pool. The value represents the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lowest of these three values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- session RUNTIMECAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- user RUNTIMECAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- target pool RUNTIMECAP</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>BOOLEAN</td>
<td>True, if the query successfully moved to the target pool.</td>
</tr>
<tr>
<td>RESULT_REASON</td>
<td>VARCHAR</td>
<td>States reason for success or failure of the move.</td>
</tr>
</tbody>
</table>

See Also

- QUERY_PROFILES
- RESOURCE_POOL_STATUS
- RESOURCE_POOLS
RESOURCE_QUEUE

RESOURCE_REJECTION

RESOURCE_POOL_STATUS

Provides configuration settings of the various resource pools in the system, including internal pools. For detailed information about resource parameters, see CREATE RESOURCE POOL or ALTER RESOURCE POOL.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node for which information is provided.</td>
</tr>
<tr>
<td>POOL_OID</td>
<td>INTEGER</td>
<td>A unique numeric ID that identifies the pool and is assigned by the Vertica catalog.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>The name of the resource pool.</td>
</tr>
<tr>
<td>IS_INTERNAL</td>
<td>BOOLEAN</td>
<td>Denotes whether a pool is one of the Built-In Pools.</td>
</tr>
<tr>
<td>MEMORY_SIZE_KB</td>
<td>INTEGER</td>
<td>Value of MEMORYSIZE setting of the pool in kilobytes.</td>
</tr>
</tbody>
</table>
| MEMORY_SIZE_ACTUAL_KB | INTEGER | Current amount of memory, in kilobytes, allocated to the pool by the resource manager. The actual size can be less than specified in the DDL, if both the following conditions exist:
  - The pool has been recently altered in a running system.
  - The request to shuffle memory is pending. |
<p>| MEMORY_INUSE_KB  | INTEGER   | Amount of memory, in kilobytes, acquired by requests running against this pool. |
| GENERAL_MEMORY_BORROWED_KB | INTEGER | Amount of memory, in kilobytes, borrowed from the GENERAL pool by requests running against this pool. The sum of MEMORY_INUSE_KB and GENERAL_MEMORY_BORROWED_KB should be less than MAX_MEMORY_SIZE_KB. |</p>
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUEING_THRESHOLD_KB</td>
<td>INTEGER</td>
<td>Calculated as MAX_MEMORY_SIZE_KB * 0.95. When the amount of memory used by all requests against this resource pool exceeds the QUEUEING_THRESHOLD_KB, new requests against the pool are queued until memory becomes available.</td>
</tr>
<tr>
<td>MAX_MEMORY_SIZE_KB</td>
<td>INTEGER</td>
<td>Value, in kilobytes, of the MAXMEMORYSIZE parameter specified when defining the pool. After this threshold is reached, new requests against this pool are rejected or queued until memory becomes available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAX_MEMORY_SIZE_KB might not reflect the set MAXMEMORYSIZE parameter value if the specified value cannot be reached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If MAXMEMORYSIZE = 10G, but less than 2G is available, MAX_MEMORY_SIZE_KB will not reflect the original value in KB, but will instead reflect 2G in KB, since that is the highest value that can be reached.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ordinarily, the ALTER RESOURCE POOL and CREATE RESOURCE POOL statements provide error messages to prevent setting the MAXMEMORYSIZE to a value greater than available, but making system changes or other configuration changes can cause this scenario.</td>
</tr>
<tr>
<td>RUNNING_QUERY_COUNT</td>
<td>INTEGER</td>
<td>Number of queries actually running using this pool.</td>
</tr>
<tr>
<td>PLANNED_CONCURRENCY</td>
<td>INTEGER</td>
<td>Value of PLANNEDCONCURRENCY parameter specified when defining the pool.</td>
</tr>
<tr>
<td>MAX_CONCURRENCY</td>
<td>INTEGER</td>
<td>Value of MAXCONCURRENCY parameter specified when defining the pool.</td>
</tr>
<tr>
<td>IS_STANDALONE</td>
<td>BOOLEAN</td>
<td>If the pool is configured to have MEMORYSIZE equal to MAXMEMORYSIZE, the pool is considered standalone because it does not borrow any memory from the General pool.</td>
</tr>
</tbody>
</table>
| QUEUE_INTERVAL              | INTERVAL  | The interval that the request waits for resources to
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEOUT</td>
<td></td>
<td>become available before being rejected. If you set this value to NONE, Vertica displays it as NULL.</td>
</tr>
<tr>
<td>QUEUE_TIMEOUT_IN_SECONDS</td>
<td>INTEGER</td>
<td>Value of QUEUETIMEOUT parameter that was specified when defining the pool. If you set QUEUE_TIMEOUT to NONE, Vertica displays this value as NULL.</td>
</tr>
<tr>
<td>EXECUTION_PARALLELISM</td>
<td>INTEGER</td>
<td>Limits the number of threads used to process any single query issued in this resource pool.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>INTEGER</td>
<td>Value of PRIORITY parameter specified when defining the pool. When set to HOLD, Vertica sets a pool's priority to -999 so the query remains queued until QUEUETIMEOUT is reached.</td>
</tr>
<tr>
<td>RUNTIME_PRIORITY</td>
<td>VARCHAR</td>
<td>Value of RUNTIME_PRIORITY specified when defining the pool.</td>
</tr>
<tr>
<td>RUNTIME_PRIORITY_THRESHOLD</td>
<td>INTEGER</td>
<td>Value of RUNTIME_PRIORITY_THRESHOLD specified when defining the pool.</td>
</tr>
<tr>
<td>SINGLE_INITIATOR</td>
<td>BOOLEAN</td>
<td>Value of SINGLEINITIATOR parameter specified when defining the pool.</td>
</tr>
<tr>
<td>QUERY_BUDGET_KB</td>
<td>INTEGER</td>
<td>The current amount of memory that queries are tuned to use. The calculation that Vertica uses to determine this value is described in Target Memory Determination for Queries in Concurrent Environments.</td>
</tr>
<tr>
<td>Note: The calculated value can change when one or more running queries needs more than the budgeted amount to run. For a detailed example of query budget calculations, see Do You Need to Put Your Query on a Budget? in the Vertica User Community.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU_AFFINITY_SET</td>
<td>VARCHAR</td>
<td>The set of CPUs on which queries associated with this pool are executed. Can be:</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A percentage of CPUs on the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A zero-based list of CPUs (a four-CPU system c of CPUs 0, 1, 2, and 3).</td>
</tr>
<tr>
<td>CPU_AFFINITY_MASK</td>
<td>VARCHAR</td>
<td>The bit mask of CPUs available for use in this pool. <strong>Valid values</strong>: Hex-encoded binaries for the CPUs on the system. Read from right to left. <strong>Example</strong>: A pool that has exclusive use of CPUs 0 and 1 on a four-CPU system is represented by the hex value of 3, which translates to 0011. The 1’s indicates that CPU 0 and 1 are available. If no other resource pools use exclusive CPU affinities, the remaining resource pools in this example have a mask of 3, which translates to 1100. Thus, CPUs 2 and 3 of the four CPUs on the system are available to these resource pools.</td>
</tr>
<tr>
<td>CPU_AFFINITY_MODE</td>
<td>VARCHAR</td>
<td>The mode for the CPU affinity, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ANY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXCLUSIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SHARED</td>
</tr>
</tbody>
</table>

**See Also**

- CREATE RESOURCE POOL
- RESOURCE_ACQUISITIONS
- RESOURCE_POOLS
- RESOURCE_QUEUES
- RESOURCE_REJECTIONS
Managing Workloads

Using Queries to Monitor Resource Pool Size and Usage

RESOURCE_QUEUES

Provides information about requests pending for various resource pools.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The name of the node for which information is listed.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Transaction identifier for this request</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the Currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>The name of the resource pool</td>
</tr>
<tr>
<td>MEMORY_REQUESTED_KB</td>
<td>INTEGER</td>
<td>Amount of memory in kilobytes requested by this request</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>INTEGER</td>
<td>Value of PRIORITY parameter specified when defining the pool.</td>
</tr>
<tr>
<td>POSITION_IN_QUEUE</td>
<td>INTEGER</td>
<td>Position of this request within the pool’s queue</td>
</tr>
<tr>
<td>QUEUE_ENTRY_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Timestamp when the request was queued</td>
</tr>
</tbody>
</table>

See Also

- RESOURCE_ACQUISITIONS
- RESOURCE_POOLS
- RESOURCE_REJECTIONS
RESOURCE_REJECTIONDETAILS

Records an entry for each resource request that Vertica denies. This is useful for determining if there are resource space issues, as well as which users/pools encounter problems.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REJECTED_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica rejected the resource.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>REQUEST_ID</td>
<td>INTEGER</td>
<td>Unique identifier of the query request in the user session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID, STATEMENT_ID, and REQUEST_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>POOL_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the resource pool.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>Name of the resource pool</td>
</tr>
<tr>
<td>REASON</td>
<td>VARCHAR</td>
<td>Reason for rejecting this request; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Usage of single request exceeds high limit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Timed out waiting for resource reservation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Canceled waiting for resource reservation</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RESOURCE_TYPE</td>
<td>VARCHAR</td>
<td>Memory, threads, file handles or execution slots. The following list shows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the resources that are limited by the resource manager. A query might need</td>
</tr>
<tr>
<td></td>
<td></td>
<td>some amount of each resource, and if the amount needed is not available,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the query is queued and could eventually time out of the queue and be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rejected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of running plans</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of running plans on initiator node (local)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of requested threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of requested file handles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of requested KB of memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of requested KB of address space</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Execution slots are determined by MAXCONCURRENCY parameter.</td>
</tr>
</tbody>
</table>

| REJECTED_VALUE | INTEGER   | Amount of the specific resource requested by the last rejection              |

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**See Also**

RESOURCE_REJECTIONS

**RESOURCE_REJECTIONS**

Monitors requests for resources that are rejected by the Resource Manager. Information is valid only as long as the node is up and the counters reset to 0 upon node restart.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>POOL_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog that identifies the resource pool.</td>
</tr>
<tr>
<td>POOL_NAME</td>
<td>VARCHAR</td>
<td>The name of the resource pool.</td>
</tr>
<tr>
<td>REASON</td>
<td>VARCHAR</td>
<td>The reason for rejecting this request; for example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Usage of single request exceeds high limit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Timed out waiting for resource reservation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Canceled waiting for resource reservation</td>
</tr>
<tr>
<td>RESOURCE_TYPE</td>
<td>VARCHAR</td>
<td>Memory, threads, file handles or execution slots.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following list shows the resources that are limited by the resource manager. A query might need some amount of each resource, and if the amount needed is not available, the query is queued and could eventually time out of the queue and be rejected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of running plans</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of running plans on initiator node (local)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of requested threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of requested file handles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of requested KB of memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Number of requested KB of address space</td>
</tr>
<tr>
<td>REJECTION_COUNT</td>
<td>INTEGER</td>
<td>Number of requests rejected due to specified</td>
</tr>
</tbody>
</table>

Note: Execution slots are determined by MAXCONCURRENCY parameter.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason and RESOURCE_TYPE.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIRST_REJECTED_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>The time of the first rejection for this pool</td>
</tr>
<tr>
<td>Reasons for Rejection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage of single request exceeds high limit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timed out waiting for resource reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canceled waiting for resource reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST_REJECTED_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>The time of the last rejection for this pool</td>
</tr>
<tr>
<td>LAST_REJECTED_VALUE</td>
<td>INTEGER</td>
<td>The amount of the specific resource requested by the last rejection</td>
</tr>
</tbody>
</table>

Example

```sql
=> SELECT node_name, pool_name, reason, resource_type,
       rejection_count AS count,
       lastRejected_value AS value
FROM resource_rejections;
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>pool_name</th>
<th>reason</th>
<th>resource_type</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiator</td>
<td>alsohasse</td>
<td>Request exceeded high limit</td>
<td>Memory(KB)</td>
<td>1</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>ceo</td>
<td>Timedout waiting for resource request</td>
<td>Memory(KB)</td>
<td>1</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>empty</td>
<td>Request exceeded high limit</td>
<td>Queries</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>general</td>
<td>Request exceeded high limit</td>
<td>Address space(KB)</td>
<td>2</td>
</tr>
<tr>
<td>45035996273704970</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>general</td>
<td>Request exceeded high limit</td>
<td>Memory(KB)</td>
<td>24</td>
</tr>
<tr>
<td>8395584</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>sa</td>
<td>Request exceeded high limit</td>
<td>Memory(KB)</td>
<td>3</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>sa</td>
<td>Timedout waiting for resource request</td>
<td>Memory(KB)</td>
<td>1</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>small</td>
<td>Request exceeded high limit</td>
<td>Memory(KB)</td>
<td>26</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>small</td>
<td>Timedout waiting for resource request</td>
<td>Memory(KB)</td>
<td>2</td>
</tr>
<tr>
<td>122880</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initiator</td>
<td>sysdata</td>
<td>Request exceeded high limit</td>
<td>Memory(KB)</td>
<td>5</td>
</tr>
<tr>
<td>102400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(10 rows)

The following command returns the type of resources currently running on the node:
See Also

- **CLEAR_RESOURCE_REJECTIONS**
- **DISK_RESOURCE_REJECTIONS**

**RESOURCE_USAGE**

Monitors system resource management on each node.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>REQUEST_COUNT</td>
<td>INTEGER</td>
<td>The cumulative number of requests for threads, file handles, and memory (in kilobytes).</td>
</tr>
<tr>
<td>LOCAL_REQUEST_COUNT</td>
<td>INTEGER</td>
<td>The cumulative number of local requests.</td>
</tr>
<tr>
<td>REQUEST_QUEUE_DEPTH</td>
<td>INTEGER</td>
<td>The current request queue depth.</td>
</tr>
<tr>
<td>ACTIVE_THREAD_COUNT</td>
<td>INTEGER</td>
<td>The current number of active threads.</td>
</tr>
<tr>
<td>OPEN_FILE_HANDLE_COUNT</td>
<td>INTEGER</td>
<td>The current number of open file handles.</td>
</tr>
<tr>
<td>MEMORY_REQUESTED_KB</td>
<td>INTEGER</td>
<td>The memory requested in kilobytes.</td>
</tr>
<tr>
<td>ADDRESS_SPACE_REQUESTED_KB</td>
<td>INTEGER</td>
<td>The address space requested in kilobytes.</td>
</tr>
<tr>
<td>WOS_USED_BYTES</td>
<td>INTEGER</td>
<td>The size of the WOS in bytes.</td>
</tr>
<tr>
<td>WOS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in the WOS.</td>
</tr>
</tbody>
</table>
## SESSION_MARS_STORE

Shows Multiple Active Result Sets (MARS) storage information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier of the Vertica session. This identifier is unique within the cluster for the current session but can be reused in a subsequent session.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The username used to create the connection.</td>
</tr>
<tr>
<td>RESULTSET_ID</td>
<td>INTEGER</td>
<td>Identifier assigned to the result set.</td>
</tr>
<tr>
<td>ROW_COUNT</td>
<td>INTEGER</td>
<td>Number of rows requested by the query.</td>
</tr>
</tbody>
</table>
### Session Parameters

Provides information about current parameters that are configurable at the session level. To view parameters configurable at all levels, see the `CONFIGURATION_PARAMETERS` system table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMAINING_ROW_COUNT</td>
<td>INTEGER</td>
<td>Number of rows that still need to be returned.</td>
</tr>
<tr>
<td>BYTES_USED</td>
<td>INTEGER</td>
<td>The number of bytes requested.</td>
</tr>
</tbody>
</table>

### See Also

*Configuration Parameters*

### Session Profiles

Provides basic session parameters and lock time out data. To obtain information about sessions, see *Profiling Database Performance*.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>The name used to log in to the database or NULL if the session is internal.</td>
</tr>
<tr>
<td>CLIENT_HOSTNAME</td>
<td>VARCHAR</td>
<td>The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal.</td>
</tr>
<tr>
<td>LOGIN_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The date and time the user logged into the database or when the internal session was created. This field is useful for identifying sessions that have been left open for a period of time and could be idle.</td>
</tr>
<tr>
<td>LOGOUT_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The date and time the user logged out of the database or when the internal session was closed.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the session for which profiling information is captured. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>EXECUTED_STATEMENT_SUCCESS_COUNT</td>
<td>INTEGER</td>
<td>The number of successfully run statements.</td>
</tr>
<tr>
<td>EXECUTED_STATEMENT_FAILURE_COUNT</td>
<td>INTEGER</td>
<td>The number of unsuccessfully run statements.</td>
</tr>
<tr>
<td>LOCK_GRANT_COUNT</td>
<td>INTEGER</td>
<td>The number of locks granted during the session.</td>
</tr>
<tr>
<td>DEADLOCK_COUNT</td>
<td>INTEGER</td>
<td>The number of deadlocks encountered during the session.</td>
</tr>
<tr>
<td>LOCK_TIMEOUT_COUNT</td>
<td>INTEGER</td>
<td>The number of times a lock timed out during the session.</td>
</tr>
<tr>
<td>LOCK_CANCELLATION_COUNT</td>
<td>INTEGER</td>
<td>The number of times a lock was canceled during the session.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOCK_REJECTION_COUNT</td>
<td>INTEGER</td>
<td>The number of times a lock was rejected during a session.</td>
</tr>
<tr>
<td>LOCK_ERROR_COUNT</td>
<td>INTEGER</td>
<td>The number of lock errors encountered during the session.</td>
</tr>
<tr>
<td>CLIENT_TYPE</td>
<td>VARCHAR</td>
<td>The type of client from which the connection was made. Possible client type values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ADO.NET Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ODBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JDBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• vsql</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td>Returns the client version.</td>
</tr>
<tr>
<td>CLIENT_OS</td>
<td>VARCHAR</td>
<td>Returns the client operating system.</td>
</tr>
<tr>
<td>CLIENT_OS_USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.</td>
</tr>
</tbody>
</table>

**See Also**

LOCKS

**SESSION_SUBSCRIPTIONS**

Lists the session subscriptions in the database. Shows the session’s view of the node subscriptions. Session subscriptions are used when running queries within the session.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_OID</td>
<td>INTEGER</td>
<td>The OID of the node.</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
NODE_NAME | VARCHAR | The name of the subscribed node.
SHARD_OID | INTEGER | The OID of the shard.
SHARD_NAME | VARCHAR | The name of the shard the node is subscribed to.
IS_PARTICIPATING | BOOLEAN | Defines whether or not the subscription is participating in the session.

```sql
=> SELECT * FROM SESSION_SUBSCRIPTIONS;

| node_oid    | 45035996273704978 |
| node_name   | v_eon_db_node0001 |
| shard_oid   | 45035996273704980 |
| shard_name  | replica           |
| is_participating | t                 |
```

### SESSIONS

Monitors external sessions. Use this table to perform the following tasks:

- Identify users who are running lengthy queries.
- Identify users who hold locks because of an idle but uncommitted transaction.
- Determine the details of the database security used for a particular session, either Secure Socket Layer (SSL) or client authentication.
- Identify client-specific information, such as client version.

**Note:** During session initialization and termination, you might see sessions running only on nodes other than the node on which you ran the virtual table query. This is a temporary situation that corrects itself when session initialization and termination complete.

| Column Name | Data Type | Description
--- | --- | ---
NODE_NAME | VARCHAR | The node name for which information is listed.
USER_NAME | VARCHAR | The name used to log in to the database or NULL if the session is internal.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT_HOSTNAME</td>
<td>VARCHAR</td>
<td>The host name and port of the TCP socket from which the client connection was made; NULL if the session is internal. Vertica accepts either IPv4 or IPv6 connections from a client machine. If the client machine contains mappings for both IPv4 and IPv6, the server randomly chooses one IP address family to make a connection. This can cause the CLIENT_HOSTNAME column to display either IPv4 or IPv6 values, based on which address family the server chooses.</td>
</tr>
<tr>
<td>CLIENT_PID</td>
<td>INTEGER</td>
<td>The process identifier of the client process that issued this connection. Remember that the client process could be on a different machine than the server.</td>
</tr>
<tr>
<td>LOGIN_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>The date and time the user logged into the database or when the internal session was created. This field can help you identify sessions that have been left open for a period of time and could be idle.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>The identifier required to close or interrupt a session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>IDLE_SESSION_TIMEOUT</td>
<td>VARCHAR</td>
<td>Specifies how long this session can remain idle before timing out, set by <code>SET SESSION IDLESESSIONTIMEOUT</code>.</td>
</tr>
<tr>
<td>GRACE_PERIOD</td>
<td>VARCHAR</td>
<td>Specifies how long a session socket remains blocked while awaiting client input or output for a given query, set by <code>SET SESSION GRACEPERIOD</code>. If the socket is blocked for a continuous period that exceeds the grace period setting, the server shuts down the socket and throws a fatal error. The session is then terminated.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLIENT_LABEL</td>
<td>VARCHAR</td>
<td>A user-specified label for the client connection that can be set when using ODBC. See Label in Data Source Name (DSN) Connection Properties in Connecting to Vertica. An MC output value means there is a client connection to an MC-managed database for that USER_NAME.</td>
</tr>
<tr>
<td>TRANSACTION_START</td>
<td>DATE</td>
<td>The date/time the current transaction started or NULL if no transaction is running.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>A string containing the hexadecimal representation of the transaction ID, if any; otherwise, NULL.</td>
</tr>
<tr>
<td>TRANSACTION_DESCRIPTION</td>
<td>VARCHAR</td>
<td>Description of the current transaction.</td>
</tr>
<tr>
<td>STATEMENT_START</td>
<td>TIMESTAMP</td>
<td>The timestamp the current statement started execution, or NULL if no statement is running.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the currently-executing statement. A value of NULL indicates that no statement is currently being processed.</td>
</tr>
<tr>
<td>LAST_STATEMENT_DURATION_US</td>
<td>INTEGER</td>
<td>The duration of the last completed statement in microseconds.</td>
</tr>
<tr>
<td>RUNTIME_PRIORITY</td>
<td>VARCHAR</td>
<td>Specifies how many run-time resources (CPU, I/O bandwidth) are allocated to queries that are running in the resource pool.</td>
</tr>
<tr>
<td>CURRENT_STATEMENT</td>
<td>VARCHAR</td>
<td>The currently executing statement, if any. NULL indicates that no statement is currently being processed.</td>
</tr>
<tr>
<td>LAST_STATEMENT</td>
<td>VARCHAR</td>
<td>NULL if the user has just logged in; otherwise the currently running statement or the most recently completed statement.</td>
</tr>
<tr>
<td>SSL_STATE</td>
<td>VARCHAR</td>
<td>Indicates if Vertica used Secure Socket Layer (SSL).</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AUTHENTICATION_METHOD</td>
<td>VARCHAR</td>
<td>The type of client authentication used for a particular session, if known. Possible values are: \n  • Unknown \n  • Trust \n  • Reject \n  • Hash \n  • Ident \n  • LDAP \n  • GSS \n  • TLS \n  See <a href="#">Security and Authentication</a> and <a href="#">TLS/SSL Server Authentication</a>.</td>
</tr>
<tr>
<td>CLIENT_TYPE</td>
<td>VARCHAR</td>
<td>The type of client from which the connection was made. Possible client type values: \n  • ADO.NET Driver \n  • ODBC Driver \n  • JDBC Driver</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• vsql</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td>Client version.</td>
</tr>
<tr>
<td>CLIENT_OS</td>
<td>VARCHAR</td>
<td>Client operating system.</td>
</tr>
<tr>
<td>CLIENT_OS_USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.</td>
</tr>
<tr>
<td>CLIENT_AUTHENTICATION_NAME</td>
<td>VARCHAR</td>
<td>User-assigned name of the authentication method.</td>
</tr>
<tr>
<td>CLIENT_AUTHENTICATION</td>
<td>INTEGER</td>
<td>Object identifier of the client authentication method.</td>
</tr>
<tr>
<td>REQUESTED_PROTOCOL</td>
<td>INTEGER</td>
<td>The requested protocol to be used when connecting.</td>
</tr>
<tr>
<td>EFFECTIVE_PROTOCOL</td>
<td>INTEGER</td>
<td>The protocol used when connecting.</td>
</tr>
<tr>
<td>EXTERNAL_MEMORY_KB</td>
<td>INTEGER</td>
<td>Amount of memory consumed by the Java Virtual Machines associated with the session.</td>
</tr>
</tbody>
</table>

**Privileges**

A superuser has unrestricted access to all session information. Users can view information only about their own, current sessions.

**See Also**

- `CLOSE_SESSION`
- `CLOSE_ALL_SESSIONS`

**STORAGE_CONTAINERS**

Monitors information about WOS and ROS storage containers in the database.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME*</td>
<td>VARCHAR</td>
<td>Node name for which information is listed.</td>
</tr>
<tr>
<td>SCHEMA_NAME*</td>
<td>VARCHAR</td>
<td>Schema name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_ID*</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME*</td>
<td>VARCHAR</td>
<td>Projection name for which information is listed on that node.</td>
</tr>
<tr>
<td>STORAGE_TYPE*</td>
<td>VARCHAR</td>
<td>Type of storage container: ROS or WOS.</td>
</tr>
<tr>
<td>STORAGE_OID*</td>
<td>INTEGER</td>
<td>Numeric ID assigned by the Vertica catalog, which identifies the storage.</td>
</tr>
<tr>
<td>SAL_STORAGE_ID</td>
<td>VARCHAR</td>
<td>Unique hexadecimal numeric ID assigned by the Vertica catalog, which identifies the storage.</td>
</tr>
<tr>
<td>TOTAL_ROW_COUNT*</td>
<td>VARCHAR</td>
<td>Total rows in the storage container listed for that projection.</td>
</tr>
<tr>
<td>DELETED_ROW_COUNT*</td>
<td>INTEGER</td>
<td>Total rows in the storage container deleted for that projection.</td>
</tr>
<tr>
<td>USED_BYTES*</td>
<td>INTEGER</td>
<td>Total bytes in the storage container listed for that projection.</td>
</tr>
<tr>
<td>START_EPOCH*</td>
<td>INTEGER</td>
<td>Number of the start epoch in the storage container for which information is listed.</td>
</tr>
<tr>
<td>END_EPOCH*</td>
<td>INTEGER</td>
<td>Number of the end epoch in the storage container for which information is listed.</td>
</tr>
<tr>
<td>GROUPING</td>
<td>VARCHAR</td>
<td>The group by which columns are stored:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ALL: All columns are grouped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PROJECTION: Columns grouped according to projection definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- NONE: No columns grouped, despite grouping in the projection definition</td>
</tr>
</tbody>
</table>

* Column values cached for faster query performance
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• OTHER: Some grouping but neither all nor according to projection (e.g., results from add column)</td>
</tr>
<tr>
<td>SEGMENT_LOWER_BOUND</td>
<td>INTEGER</td>
<td>Lower bound of the segment range spanned by the storage container or NULL if the corresponding projection is not elastic.</td>
</tr>
<tr>
<td>SEGMENT_UPPER_BOUND</td>
<td>INTEGER</td>
<td>Upper bound of the segment range spanned by the storage container or NULL if the corresponding projection is not elastic.</td>
</tr>
<tr>
<td>IS_SORTED</td>
<td>BOOLEAN</td>
<td>Whether the storage container’s data is sorted (WOS containers only).</td>
</tr>
<tr>
<td>LOCATION_LABEL</td>
<td>VARCHAR (128)</td>
<td>The location label (if any) for the storage container is stored.</td>
</tr>
<tr>
<td>DELETE_VECTOR_COUNT</td>
<td>INTEGER</td>
<td>The number of delete vectors in the storage container.</td>
</tr>
</tbody>
</table>

* Column values cached for faster query performance, continued

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**STORAGE_POLICIES**

Monitors the current storage policies in effect for one or more database objects.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>Schema name for which information is listed.</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>VARCHAR</td>
<td>The name of the database object associated through the storage policy.</td>
</tr>
<tr>
<td>POLICYDETAILS</td>
<td>VARCHAR</td>
<td>The object type of the storage policy.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOCATION_LABEL</td>
<td>VARCHAR</td>
<td>The label for this storage location.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION_LABEL</td>
<td>VARCHAR</td>
<td>The label associated with a specific storage location. The storage_tiers system table includes data totals for unlabeled locations, which are considered labeled with empty strings ('').</td>
</tr>
<tr>
<td>NODE_COUNT</td>
<td>INTEGER</td>
<td>The total number of nodes that include a storage location named location_label.</td>
</tr>
<tr>
<td>LOCATION_COUNT</td>
<td>INTEGER</td>
<td>The total number of storage locations named location_label. This value can differ from node_count if you create</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
| | | labeled locations with the same name at different paths on different nodes. For example: 
| | | vVmart_node0001: Create one labeled location, FAST 
| | | VVmart_node0002: Create two labeled locations, FAST, at different directory paths 
| | | In this case, node_count value = 2, while location_count value = 3. 
| ROS_CONTAINER_COUNT | INTEGER | The total number of ROS containers stored across all cluster nodes for location_label. 
| TOTAL_OCCUPIED_SIZE | INTEGER | The total number of bytes that all ROS containers for location_label occupy across all cluster nodes. 

### Privileges
None

### See Also
- DISK_STORAGE
- STORAGE_POLICIES
- STORAGE_USAGE
- Storage Management Functions

### STORAGE_USAGE
Provides information about file system storage usage. This is useful for determining disk space usage trends.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLL_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Time when Vertica recorded the row.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>PATH</td>
<td>VARCHAR</td>
<td>Path where the storage location is mounted.</td>
</tr>
<tr>
<td>DEVICE</td>
<td>VARCHAR</td>
<td>Device on which the storage location is mounted.</td>
</tr>
<tr>
<td>FILESYSTEM</td>
<td>VARCHAR</td>
<td>Filesystem on which the storage location is mounted.</td>
</tr>
<tr>
<td>USED_BYTES</td>
<td>INTEGER</td>
<td>Counter history of number of used bytes.</td>
</tr>
<tr>
<td>FREE_BYTES</td>
<td>INTEGER</td>
<td>Counter history of number of free bytes.</td>
</tr>
<tr>
<td>USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Percent of storage in use.</td>
</tr>
</tbody>
</table>

Privileges

Superuser

See Also

- DISK_STORAGE
- STORAGE_CONTAINERS
- STORAGE_POLICIES
- STORAGE_TIERS
- Storage Management Functions

STRATA

Contains internal details of how the Tuple Mover combines ROS containers in each projection, broken down by stratum and classifies the ROS containers by size and partition. The related STRATA_STRUCTURES table provides a summary of the strata values.

Mergeout in the Administrator's Guide describes how the Tuple Mover combines ROS containers.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed on that node.</td>
</tr>
<tr>
<td>PARTITION_KEY</td>
<td>VARCHAR</td>
<td>References the partition or partition group for which information is listed.</td>
</tr>
<tr>
<td>STRATA_COUNT</td>
<td>INTEGER</td>
<td>The total number of strata for this projection partition.</td>
</tr>
<tr>
<td>MERGING_STRATA_COUNT</td>
<td>INTEGER</td>
<td>The number of strata the Tuple Mover can merge out.</td>
</tr>
<tr>
<td>STRATUM_CAPACITY</td>
<td>INTEGER</td>
<td>The maximum number of ROS containers for the stratum before they must be merged.</td>
</tr>
<tr>
<td>STRATUM_HEIGHT</td>
<td>FLOAT</td>
<td>The size ratio between the smallest and largest ROS container in this stratum.</td>
</tr>
<tr>
<td>STRATUM_NO</td>
<td>INTEGER</td>
<td>The stratum number. Strata are numbered starting at 0, for the stratum containing the smallest ROS containers.</td>
</tr>
<tr>
<td>STRATUM_LOWER_SIZE</td>
<td>VARCHAR</td>
<td>The smallest ROS container size allowed in this stratum.</td>
</tr>
<tr>
<td>STRATUM_UPPER_SIZE</td>
<td>VARCHAR</td>
<td>The largest ROS container size allowed in this stratum.</td>
</tr>
<tr>
<td>ROS_CONTAINER_COUNT</td>
<td>INTEGER</td>
<td>The current number of ROS containers in the projection partition.</td>
</tr>
</tbody>
</table>

**STRATA_STRUCTURES**

This table provides an overview of Tuple Mover internal details. It summarizes how the ROS containers are classified by size. A more detailed view can be found in the STRATA virtual table.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The schema name for which information is listed.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>The projection name for which information is listed on that node.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>A unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>PARTITION_KEY</td>
<td>VARCHAR</td>
<td>References the partition or partition group for which information is listed.</td>
</tr>
<tr>
<td>STRATA_COUNT</td>
<td>INTEGER</td>
<td>The total number of strata for this projection partition.</td>
</tr>
<tr>
<td>MERGING_STRATA_COUNT</td>
<td>INTEGER</td>
<td>In certain hardware configurations, a high strata could contain more ROS containers than the Tuple Mover can merge out; output from this column denotes the number of strata the Tuple Mover can merge out.</td>
</tr>
<tr>
<td>STRATUM_CAPACITY</td>
<td>INTEGER</td>
<td>The maximum number of ROS containers that the strata can contained before it must merge them.</td>
</tr>
<tr>
<td>STRATUM_HEIGHT</td>
<td>FLOAT</td>
<td>The size ratio between the smallest and largest ROS container in a stratum.</td>
</tr>
<tr>
<td>ACTIVE_STRATA_COUNT</td>
<td>INTEGER</td>
<td>The total number of strata that have ROS containers in them.</td>
</tr>
</tbody>
</table>

**Example**

```sql
=> \pset expanded
Expanded display is on.
=> SELECT node_name, schema_name, projection_name, strata_count, stratum_capacity, stratum_height, active_strata_count
    FROM strata_structures WHERE stratum_capacity > 60;

- [ RECORD 1 ]-----------------------------------------------
  node_name       | v_vmartdb_node001
  schema_name     | public
  projection_name | shipping_dimension_ODD_22_seg_vmart_design_vmart_design
  partition_key   |
  strata_count    | 4
  stratum_capacity | 62
  stratum_height  | 25.6511590887058
```
<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmartdb_node001</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>projection_name</td>
<td>shipping_dimension_OBD_23_seg_vmart_design_vmart_design</td>
</tr>
<tr>
<td>partition_key</td>
<td></td>
</tr>
<tr>
<td>strata_count</td>
<td>4</td>
</tr>
<tr>
<td>stratum_capacity</td>
<td>62</td>
</tr>
<tr>
<td>stratum_height</td>
<td>25.6511590887058</td>
</tr>
<tr>
<td>active_strata_count</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmartdb_node002</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>projection_name</td>
<td>shipping_dimension_OBD_22_seg_vmart_design_vmart_design</td>
</tr>
<tr>
<td>partition_key</td>
<td></td>
</tr>
<tr>
<td>strata_count</td>
<td>4</td>
</tr>
<tr>
<td>stratum_capacity</td>
<td>62</td>
</tr>
<tr>
<td>stratum_height</td>
<td>25.6511590887058</td>
</tr>
<tr>
<td>active_strata_count</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmartdb_node003</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>projection_name</td>
<td>shipping_dimension_OBD_22_seg_vmart_design_vmart_design</td>
</tr>
<tr>
<td>partition_key</td>
<td></td>
</tr>
<tr>
<td>strata_count</td>
<td>4</td>
</tr>
<tr>
<td>stratum_capacity</td>
<td>62</td>
</tr>
<tr>
<td>stratum_height</td>
<td>25.6511590887058</td>
</tr>
<tr>
<td>active_strata_count</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node_name</th>
<th>v_vmartdb_node004</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema_name</td>
<td>public</td>
</tr>
<tr>
<td>projection_name</td>
<td>shipping_dimension_OBD_22_seg_vmart_design_vmart_design</td>
</tr>
<tr>
<td>partition_key</td>
<td></td>
</tr>
<tr>
<td>strata_count</td>
<td>4</td>
</tr>
<tr>
<td>stratum_capacity</td>
<td>62</td>
</tr>
<tr>
<td>stratum_height</td>
<td>25.6511590887058</td>
</tr>
<tr>
<td>active_strata_count</td>
<td>1</td>
</tr>
</tbody>
</table>
**SYSTEM**

Monitors the overall state of the database.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_EPOCH</td>
<td>INTEGER</td>
<td>The current epoch number.</td>
</tr>
<tr>
<td>AHM_EPOCH</td>
<td>INTEGER</td>
<td>The AHM epoch number.</td>
</tr>
<tr>
<td>LAST_GOOD_EPOCH</td>
<td>INTEGER</td>
<td>The smallest (min) of all the checkpoint epochs on the cluster.</td>
</tr>
<tr>
<td>REFRESH_EPOCH</td>
<td>INTEGER</td>
<td>Deprecated, always set to -1.</td>
</tr>
<tr>
<td>DESIGNED_FAULT_TOLERANCE</td>
<td>INTEGER</td>
<td>The designed or intended K-safety level.</td>
</tr>
<tr>
<td>NODE_COUNT</td>
<td>INTEGER</td>
<td>The number of nodes in the cluster.</td>
</tr>
<tr>
<td>NODE_DOWN_COUNT</td>
<td>INTEGER</td>
<td>The number of nodes in the cluster that are currently down.</td>
</tr>
<tr>
<td>CURRENT_FAULT_TOLERANCE</td>
<td>INTEGER</td>
<td>The number of node failures the cluster can tolerate before it shuts down automatically. This is the current K-safety level.</td>
</tr>
<tr>
<td>CATALOG_REVISION_NUMBER</td>
<td>INTEGER</td>
<td>The catalog version number.</td>
</tr>
<tr>
<td>WOS_USED_BYTES</td>
<td>INTEGER</td>
<td>The WOS size in bytes (cluster-wide).</td>
</tr>
<tr>
<td>WOS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in WOS (cluster-wide).</td>
</tr>
<tr>
<td>ROS_USED_BYTES</td>
<td>INTEGER</td>
<td>The ROS size in bytes (cluster-wide).</td>
</tr>
<tr>
<td>ROS_ROW_COUNT</td>
<td>INTEGER</td>
<td>The number of rows in ROS (cluster-wide).</td>
</tr>
<tr>
<td>TOTAL_USED_BYTES</td>
<td>INTEGER</td>
<td>The total storage in bytes (WOS + ROS) (cluster-wide).</td>
</tr>
</tbody>
</table>
### SYSTEM_Resource_usage

Provides history about system resources, such as memory, CPU, network, disk, I/O.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL_ROW_COUNT</td>
<td>INTEGER</td>
<td>The total number of rows (WOS + ROS) (cluster-wide).</td>
</tr>
<tr>
<td>SYSTEMRESOURCE_USAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>End time of the history interval.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>AVERAGE_MEMORY_USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Average memory usage in percent of total memory (0-100) during the history interval.</td>
</tr>
<tr>
<td>AVERAGE_CPU_USAGE_PERCENT</td>
<td>FLOAT</td>
<td>Average CPU usage in percent of total CPU time (0-100) during the history interval.</td>
</tr>
<tr>
<td>NET_RX_KBYTES_PER_SECOND</td>
<td>FLOAT</td>
<td>Average number of kilobytes received from network (incoming) per second during the history interval.</td>
</tr>
<tr>
<td>NET_TX_KBYTES_PER_SECOND</td>
<td>FLOAT</td>
<td>Average number of kilobytes transmitting to network (outgoing) per second during the history interval.</td>
</tr>
<tr>
<td>IO_READ_KBYTES_PER_SECOND</td>
<td>FLOAT</td>
<td>Disk I/O average number of kilobytes read from disk per second during the history interval.</td>
</tr>
<tr>
<td>IO_WRITTEN_KBYTES_PER_SECOND</td>
<td>FLOAT</td>
<td>Average number of kilobytes written to disk per second during the history interval.</td>
</tr>
</tbody>
</table>

### Privileges

Superuser
SYSTEM_SERVICES

Provides information about background system services that the Workload Analyzer monitors.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>SERVICE_TYPE</td>
<td>VARCHAR</td>
<td>Type of service; can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SYSTEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TUPLE MOVER</td>
</tr>
<tr>
<td>SERVICE_GROUP</td>
<td>VARCHAR</td>
<td>Group name, if there are multiple services of the same type.</td>
</tr>
<tr>
<td>SERVICE_NAME</td>
<td>VARCHAR</td>
<td>Name of the service.</td>
</tr>
<tr>
<td>SERVICE_INTERVAL_SEC</td>
<td>INTEGER</td>
<td>How often the service is executed (in seconds) during the history interval.</td>
</tr>
<tr>
<td>IS_ENABLED</td>
<td>BOOLEAN</td>
<td>Denotes if the service is enabled.</td>
</tr>
<tr>
<td>LAST_RUN_START</td>
<td>TIMESTAMPTZ</td>
<td>Denotes when the service was started last time.</td>
</tr>
<tr>
<td>LAST_RUN_END</td>
<td>TIMESTAMPTZ</td>
<td>Denotes when the service was completed last time.</td>
</tr>
</tbody>
</table>

Privileges

No explicit privileges are required. You only see the records for tables that you have privileges to view.

SYSTEM_SESSIONS

Provides information about system internal session history by system task.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>INTEGER</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>VARCHAR</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>SESSION_TYPE</td>
<td>VARCHAR</td>
<td>Session type. Can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CLIENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DBD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MERGEOUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MOVEOUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REBALANCE_CLUSTERS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RECOVERY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REFRESH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIMER_SERVICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CONNECTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SUBSESSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• REPARTITION_TABLE</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LICENSE_AUDIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTUP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VSPREAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUNTIME_PRIORITY</td>
<td>VARCHAR</td>
<td>Specifies how many run-time resources (CPU, I/O bandwidth) are allocated to queries that are running in the resource pool.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>Transaction description in this session.</td>
</tr>
<tr>
<td>SESSION_START_TTIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Value of session at beginning of history interval.</td>
</tr>
<tr>
<td>SESSION_END_TTIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Value of session at end of history interval.</td>
</tr>
<tr>
<td>IS_ACTIVE</td>
<td>BOOLEAN</td>
<td>Denotes if the session is still running.</td>
</tr>
<tr>
<td>SESSION_DURATIOM_MS</td>
<td>INTEGER</td>
<td>Duration of the session in milliseconds.</td>
</tr>
<tr>
<td>CLIENT_TYPE</td>
<td>VARCHAR</td>
<td>Columns not used in SYSTEM_SESSIONS system table. To view values for these columns, see the V_MONITOR schema system tables SESSIONS, USER_SESSIONS, CURRENT_SESSION, and SESSION_PROFILES.</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>CLIENT_OS</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>CLIENT_OS_USERNAME</td>
<td>VARCHAR</td>
<td>The name of the user that logged into, or attempted to log into, the database. This is logged even when the login attempt is unsuccessful.</td>
</tr>
</tbody>
</table>

**Privileges**

Superuser
**TABLE_RECOVERIES**

Provides detailed information about recovered and recovering tables during a recovery by table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is performing the recovery.</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR</td>
<td>The status of the table. Tables can have the following status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recovered - The table is fully recovered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• recovering - The table is in the process of recovery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• error_retry - Vertica has attempted to recover the table, but the recovery failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tables that have not yet begun the recovery process do not have a status.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the table being recovered.</td>
</tr>
<tr>
<td>TABLE_OID</td>
<td>INTEGER</td>
<td>The object ID of the table being recovered.</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMPZ</td>
<td>The date and time that the table began recovery.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMPZ</td>
<td>The date and time that the table completed recovery.</td>
</tr>
<tr>
<td>PHASE</td>
<td>VARCHAR</td>
<td>The phase of the recovery.</td>
</tr>
<tr>
<td>RECOVER_PRIORITY</td>
<td>INTEGER</td>
<td>The recovery priority of the table being recovered.</td>
</tr>
<tr>
<td>THREAD_ID</td>
<td>VARCHAR</td>
<td>The ID of the thread that performed the recovery.</td>
</tr>
</tbody>
</table>

**Privileges**

None
TABLE_RECOVERY_STATUS

Provides node recovery information during a Recovery By Table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is performing the recovery.</td>
</tr>
<tr>
<td>NODE_RECOVERY_START_TIME</td>
<td>TIMESTAMPZ</td>
<td>The timestamp for when the node began recovering.</td>
</tr>
<tr>
<td>IS_RUNNING</td>
<td>BOOLEAN</td>
<td>Indicates if the recovery process is still running.</td>
</tr>
<tr>
<td>RECOVER_EPOCH</td>
<td>INTEGER</td>
<td>The epoch that the recovery operation is trying to recover to.</td>
</tr>
<tr>
<td>RECOVERING_TABLE_NAME</td>
<td>VARCHAR</td>
<td>The name of the table currently recovering.</td>
</tr>
<tr>
<td>RECOVERED_TABLE_COUNT</td>
<td>INTEGER</td>
<td>Indicates how many tables on the node have already recovered.</td>
</tr>
<tr>
<td>TABLE_COUNT</td>
<td>INTEGER</td>
<td>The total number of tables on the node.</td>
</tr>
</tbody>
</table>

Privileges

None

TRANSACTIONS

Records the details of each transaction.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Beginning of history interval.</td>
</tr>
<tr>
<td>END_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>End of history interval.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the user.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user for which transaction information is listed.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>Textual description of the transaction.</td>
</tr>
<tr>
<td>START_EPOCH</td>
<td>INTEGER</td>
<td>Number of the start epoch for the transaction.</td>
</tr>
<tr>
<td>END_EPOCH</td>
<td>INTEGER</td>
<td>Number of the end epoch for the transaction</td>
</tr>
<tr>
<td>NUMBER_OF_STATEMENTS</td>
<td>INTEGER</td>
<td>Number of query statements executed in this transaction.</td>
</tr>
<tr>
<td>ISOLATION</td>
<td>VARCHAR</td>
<td>Denotes the transaction mode as &quot;READ COMMITTED&quot; or &quot;SERIALIZABLE&quot;.</td>
</tr>
<tr>
<td>IS_READ_ONLY</td>
<td>BOOLEAN</td>
<td>Denotes &quot;READ ONLY&quot; transaction mode.</td>
</tr>
<tr>
<td>IS_COMMITTED</td>
<td>BOOLEAN</td>
<td>Determines if the transaction was committed. False means ROLLBACK.</td>
</tr>
<tr>
<td>IS_LOCAL</td>
<td>BOOLEAN</td>
<td>Denotes transaction is local (non-distributed).</td>
</tr>
<tr>
<td>IS_INITIATOR</td>
<td>BOOLEAN</td>
<td>Denotes if the transaction occurred on this node (t).</td>
</tr>
<tr>
<td>IS_DDL</td>
<td>BOOLEAN</td>
<td>Distinguishes between a DDL transaction (t) and non-DDL transaction (f).</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.
See Also

Transactions

TRUNCATED_SCHEMA

Lists the original names of restored schemas that were truncated due to name lengths exceeding 128 characters.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE_TIME</td>
<td>TIMESTAMPZ</td>
<td>The time that the table was restored.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for the restoring session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>USER_ID</td>
<td>INTEGER</td>
<td>Identifier of the user for the restore event.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user for which Vertica lists restore information at the time it recorded the session.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any; otherwise NULL.</td>
</tr>
<tr>
<td>ORIGINAL_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The original name of the schema prior to the restore.</td>
</tr>
<tr>
<td>NEW_SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema after it was truncated.</td>
</tr>
</tbody>
</table>

Privileges

None

TUNING_RECOMMENDATIONS

Returns tuning recommendation results from the last ANALYZE_WORKLOAD call. This information is useful for letting you build filters on the Workload Analyzer result set.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>observation_count</td>
<td>INTEGER</td>
<td>Integer for the total number of events observed for this tuning recommendation. For example, if you see a return value of 1, WLA is making its first tuning recommendation for the event in 'scope'.</td>
</tr>
<tr>
<td>first_observation_time</td>
<td>TIMESTAMPTZ</td>
<td>Timestamp when the event first occurred. If this column returns a null value, the tuning recommendation is from the current status of the system instead of from any prior event.</td>
</tr>
<tr>
<td>last_observation_time</td>
<td>TIMESTAMPTZ</td>
<td>Timestamp when the event last occurred. If this column returns a null value, the tuning recommendation is from the current status of the system instead of from any prior event.</td>
</tr>
<tr>
<td>tuning_parameter</td>
<td>VARCHAR</td>
<td>Objects on which you should perform a tuning action. For example, a return value of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• public.t informs the DBA to run Database Designer on table t in the public schema</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• bsmith notifies a DBA to set a password for user bsmith</td>
</tr>
<tr>
<td>tuning_description</td>
<td>VARCHAR</td>
<td>Textual description of the tuning recommendation from the Workload Analyzer to perform on the tuning_parameter object. Examples of some of the returned values include, but are not limited to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Run database designer on table schema.table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Create replicated projection for table schema.table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Consider incremental design on query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reset configuration parameter with ALTER DATABASE dbName SET parameter = value;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Re-segment projection projection-name on high-cardinality column(s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Drop the projection projection-name</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Alter a table's partition expression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reorganize data in partitioned table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Decrease the MoveOutInterval configuration parameter setting</td>
</tr>
<tr>
<td>tuning_command</td>
<td>VARCHAR</td>
<td>Command string if tuning action is a SQL command. For example, the following example statements recommend that the DBA:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update statistics on a particular schema's table.column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT ANALYZE_STATISTICS('public.table.column');</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resolve mismatched configuration parameter 'LockTimeout':</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'LockTimeout';</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set the password for user bsmith:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALTER USER (user) IDENTIFIED BY ('new_password');</td>
</tr>
<tr>
<td>tuning_cost</td>
<td>VARCHAR</td>
<td>Cost is based on the type of tuning recommendation and is one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOW—minimal impact on resources from running the tuning command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MEDIUM—moderate impact on resources from running the tuning command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HIGH—maximum impact on resources from running the tuning command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depending on the size of your database or table, consider running high-cost operations after hours instead of during peak load times.</td>
</tr>
</tbody>
</table>

**Privileges**

**Superuser**
Examples

See `ANALYZE_WORKLOAD`.

See Also

- Analyzing Workloads
- Understanding WLA Triggering Conditions

TUPLE_MOVER_OPERATIONS

Monitors the status of Tuple Mover operations on each node.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION_START_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>Start time of a Tuple Mover operation.</td>
</tr>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Node name for which information is listed.</td>
</tr>
<tr>
<td>OPERATION_NAME</td>
<td>VARCHAR</td>
<td>One of the following operations:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Moveout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mergeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Analyze Statistics</td>
</tr>
<tr>
<td>OPERATION_STATUS</td>
<td>VARCHAR</td>
<td>Returns the status of each operation, one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Empty string: Not running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Update</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Change plan type to Replay Delete</td>
</tr>
<tr>
<td>TABLE_SCHEMA</td>
<td>VARCHAR</td>
<td>Schema name for the specified projection.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR</td>
<td>Table name for the specified projection.</td>
</tr>
<tr>
<td>PROJECTION_NAME</td>
<td>VARCHAR</td>
<td>Name of the projection being processed.</td>
</tr>
<tr>
<td>PROJECTION_ID</td>
<td>INTEGER</td>
<td>Unique numeric ID assigned by the Vertica catalog, which identifies the projection.</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>INTEGER</td>
<td>Identifier for the column for the associated projection being processed.</td>
</tr>
<tr>
<td>EARLIEST_CONTAINER_START_EPOCH</td>
<td>INTEGER</td>
<td>Populated for mergeout and purge operations only. For an automatically-invoked mergeout, for example, the returned value represents the lowest epoch of containers involved in the mergeout.</td>
</tr>
<tr>
<td>LATEST_CONTAINER_END_EPOCH</td>
<td>INTEGER</td>
<td>Populated for mergeout and purge_partitions operations. For an automatically-invoked mergeout, for example, the returned value represents the highest epoch of containers involved in the mergeout.</td>
</tr>
<tr>
<td>ROS_COUNT</td>
<td>INTEGER</td>
<td>Number of ROS containers.</td>
</tr>
<tr>
<td>TOTAL_ROS_USED_BYTES</td>
<td>INTEGER</td>
<td>Size in bytes of all ROS containers in the mergeout operation. (Not applicable for other operations.)</td>
</tr>
<tr>
<td>PLAN_TYPE</td>
<td>VARCHAR</td>
<td>One of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Moveout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mergeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Analyze Statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Replay Delete: A marker that the Tuple</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>Mover</td>
<td>sets</td>
<td>on a container during recovery, to prevent concurrent mergeout.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>INTEGER</td>
<td>Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.</td>
</tr>
<tr>
<td>IS_EXECUTING</td>
<td>BOOLEAN</td>
<td>Distinguishes between actively-running (t) and completed (f) tuple mover operations.</td>
</tr>
<tr>
<td>RUNTIME_PRIORITY</td>
<td>VARCHAR</td>
<td>Determines how many run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to running queries in the resource pool. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HIGH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MEDIUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOW</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.

**Example**

```sql
=> SELECT node_name, operation_status, projection_name, plan_type
    FROM TUPLE_MOVER_OPERATIONS;
```

```
+---------------+-------------+------------------+-------------+
<table>
<thead>
<tr>
<th>node_name</th>
<th>operation_status</th>
<th>projection_name</th>
<th>plan_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>Running</td>
<td>p1_b2</td>
<td>Mergeout</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>Running</td>
<td>p1</td>
<td>Mergeout</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>Running</td>
<td>p1_b2</td>
<td>Replay Delete</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>Running</td>
<td>p1_b2</td>
<td>Mergeout</td>
</tr>
</tbody>
</table>
```
See Also

- **DO_TM_TASK**
- **PURGE**

### UDFS_EVENTS

Records information about events involving the S3 file system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_TIME</td>
<td>TIMESTAMPTZ</td>
<td>Starting timestamp of the event.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMPTZ</td>
<td>Ending time of the event.</td>
</tr>
<tr>
<td>FILESYSTEM</td>
<td>VARCHAR</td>
<td>Name of the file system, such as S3.</td>
</tr>
<tr>
<td>PATH</td>
<td>VARCHAR</td>
<td>Complete file path.</td>
</tr>
<tr>
<td>EVENT</td>
<td>VARCHAR</td>
<td>The function call that was made, for example &quot;virtual size_t SAL::S3FileOperator::read(void*, size_t)&quot;.</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR</td>
<td>Status of the event: OK, CANCEL, or FAIL.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
<td>Other details of the event. For example, a read event reports the size of what was read.</td>
</tr>
</tbody>
</table>

### UDFS_STATISTICS

Records aggregate information about operations on the S3 file system.

An operation can be made up of many individual read, write, or retry requests. SUCCESSFUL_OPERATIONS and FAILED_OPERATIONS count operations; the other counters count individual
requests. When an operation finishes, one of the OPERATIONS counters is incremented once, but several other counters could be incremented several times each.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILESYSTEM</td>
<td>VARCHAR</td>
<td>Name of the file system, such as S3.</td>
</tr>
<tr>
<td>SUCCESSFUL_OPERATIONS</td>
<td>INTEGER</td>
<td>Number of successful file-system operations.</td>
</tr>
<tr>
<td>FAILED_OPERATIONS</td>
<td>INTEGER</td>
<td>Number of failed file-system operations.</td>
</tr>
<tr>
<td>RETRIES</td>
<td>INTEGER</td>
<td>Number of retry events.</td>
</tr>
<tr>
<td>METADATA_READS</td>
<td>INTEGER</td>
<td>Number of requests to read metadata. For example, S3 list bucket and HEAD requests are metadata reads.</td>
</tr>
<tr>
<td>METADATA_WRITES</td>
<td>INTEGER</td>
<td>Number of requests to write metadata. For example, S3 POST and DELETE requests are metadata writes.</td>
</tr>
<tr>
<td>DATA_READS</td>
<td>INTEGER</td>
<td>Number of read operations, such as S3 GET requests to download files.</td>
</tr>
<tr>
<td>DATA_WRITES</td>
<td>INTEGER</td>
<td>Number of write operations, such as S3 PUT requests to upload files.</td>
</tr>
<tr>
<td>DOWNSTREAM_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes received.</td>
</tr>
<tr>
<td>UPSTREAM_BYTES</td>
<td>INTEGER</td>
<td>Number of bytes sent.</td>
</tr>
</tbody>
</table>

**UDX_FENCED_PROCESSES**

Provides information about processes Vertica uses to run user-defined extensions in fenced mode.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>PROCESS_TYPE</td>
<td>VARCHAR</td>
<td>Indicates what kind of side process this row is for and can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UDxZygoteProcess — Master process that creates worker side processes, as needed, for queries. There</td>
</tr>
</tbody>
</table>
### Column Name | Data Type | Description
--- | --- | ---
 | | will be, at most, 1 UP UDxZygoteProcess for each Vertica instance.
 | | UDxSideProcess — Indicates that the process is a worker side process. There could be many UDxSideProcesses, depending on how many sessions there are, how many queries, and so on.

| SESSION_ID | VARCHAR | Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.
| LANGUAGE | VARCHAR | The language of the UDx. For example 'R' or 'C++';
| MAX_MEMORY_JAVA_KB | INTEGER | The maximum amount of memory in KB that can be used for the Java heap file on the node.
| PID | INTEGER | Linux process identifier of the side process (UDxSideProcess).
| PORT | VARCHAR | For Vertica internal use. The TCP port that the side process is listening on.
| STATUS | VARCHAR | Set to UP or DOWN, depending on whether the process is alive or not.

After a process fails, Vertica restarts it only on demand. So after a process failure, there might be periods of time when no side processes run.

### Privileges
None

### USER_LIBRARIES

Lists the user libraries that are currently loaded. These libraries contain user-defined extensions (UDxs) that provide additional analytic functions.
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR (8192)</td>
<td>The name of the schema containing the library.</td>
</tr>
<tr>
<td>LIB_NAME</td>
<td>VARCHAR (8192)</td>
<td>The name of the library.</td>
</tr>
<tr>
<td>LIB_OID</td>
<td>INTEGER</td>
<td>The object ID of the library.</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>VARCHAR (8192)</td>
<td>The creator of the library file.</td>
</tr>
<tr>
<td>OWNER_ID</td>
<td>INTEGER</td>
<td>The object ID of the library's owner.</td>
</tr>
<tr>
<td>LIB_FILE_NAME</td>
<td>VARCHAR (8192)</td>
<td>The name of the shared library file.</td>
</tr>
<tr>
<td>MD5_SUM</td>
<td>VARCHAR (8192)</td>
<td>The MD5 checksum of the library file, used to verify that the file was correctly copied to each node. Note: This use of MD5 is not for cryptographic or authentication purposes. For information on authenticating with MD5 see Hash Authentication.</td>
</tr>
<tr>
<td>SDK_VERSION</td>
<td>VARCHAR (8192)</td>
<td>The version of the Vertica SDK used to compile the library.</td>
</tr>
<tr>
<td>REVISION</td>
<td>VARCHAR (8192)</td>
<td>The revision of the Vertica SDK used to compile the library.</td>
</tr>
<tr>
<td>LIB_BUILD_TAG</td>
<td>VARCHAR (8192)</td>
<td>Internal information set by library developer to track the when the library was compiled.</td>
</tr>
<tr>
<td>LIB_VERSION</td>
<td>VARCHAR (8192)</td>
<td>The version of the library.</td>
</tr>
<tr>
<td>LIB_SDK_VERSION</td>
<td>VARCHAR (8192)</td>
<td>The version of the Vertica SDK intended for use with the library. The developer sets this value manually. This value may differ from the values in the SDK_VERSION and REVISION, which are set automatically during compilation.</td>
</tr>
<tr>
<td>SOURCE_URL</td>
<td>VARCHAR (8192)</td>
<td>A URL that contains information about the library.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR (8192)</td>
<td>A description of the library.</td>
</tr>
<tr>
<td>LICENSES_REQUIRED</td>
<td>VARCHAR (8192)</td>
<td>The licenses required to use the library.</td>
</tr>
<tr>
<td>SIGNATURE</td>
<td>VARCHAR (8192)</td>
<td>The signature used to sign the library for validation.</td>
</tr>
<tr>
<td>DEPENDENCIES</td>
<td>VARCHAR (8192)</td>
<td>External libraries on which this library depends. These libraries are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maintained by Vertica, just like the user libraries themselves.</td>
</tr>
</tbody>
</table>

**USER_LIBRARY_MANIFEST**

Lists user-defined functions contained in all loaded user libraries.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_NAME</td>
<td>VARCHAR</td>
<td>The name of the schema containing the function.</td>
</tr>
<tr>
<td>LIB_NAME</td>
<td>VARCHAR</td>
<td>The name of the library containing the UDF.</td>
</tr>
<tr>
<td>LIB_OID</td>
<td>INTEGER</td>
<td>The object ID of the library containing the function.</td>
</tr>
<tr>
<td>OBJ_NAME</td>
<td>VARCHAR</td>
<td>The name of the constructor class in the library for a function.</td>
</tr>
<tr>
<td>OBJ_TYPE</td>
<td>VARCHAR</td>
<td>The type of user defined function (scalar function, transform function)</td>
</tr>
<tr>
<td>ARG_TYPES</td>
<td>VARCHAR</td>
<td>A comma-delimited list of data types of the function's parameters.</td>
</tr>
<tr>
<td>RETURN_TYPE</td>
<td>VARCHAR</td>
<td>A comma-delimited list of data types of the function's return values.</td>
</tr>
</tbody>
</table>
Privileges

None

USER_SESSIONS

Returns user session history on the system.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>Name of the node that is reporting the requested information.</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR</td>
<td>Name of the user at the time Vertica recorded the session.</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>VARCHAR</td>
<td>Identifier for this session. This identifier is unique within the cluster at any point in time but can be reused when the session closes.</td>
</tr>
<tr>
<td>TRANSACTION_ID</td>
<td>VARCHAR</td>
<td>Identifier for the transaction within the session, if any. If a session is active but no transaction has begun, TRANSACTION_ID returns NULL.</td>
</tr>
<tr>
<td>STATEMENT_ID</td>
<td>VARCHAR</td>
<td>Unique numeric ID for the currently-running statement. NULL indicates that no statement is currently being processed. The combination of TRANSACTION_ID and STATEMENT_ID uniquely identifies a statement within a session.</td>
</tr>
<tr>
<td>RUNTIME_PRIORITY</td>
<td>VARCHAR</td>
<td>Determines the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to queries already running in the resource pool. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HIGH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MEDIUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LOW</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Queries with a HIGH run-time priority are given more CPU and I/O resources than those with a MEDIUM or LOW run-time priority.</td>
</tr>
<tr>
<td>SESSION_START_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Value of session at beginning of history interval.</td>
</tr>
<tr>
<td>SESSION_END_TIMESTAMP</td>
<td>TIMESTAMPTZ</td>
<td>Value of session at end of history interval.</td>
</tr>
<tr>
<td>IS_ACTIVE</td>
<td>BOOLEAN</td>
<td>Denotes if the operation is executing.</td>
</tr>
<tr>
<td>CLIENT_HOSTNAME</td>
<td>VARCHAR</td>
<td>IP address of the client system</td>
</tr>
<tr>
<td>CLIENT_PID</td>
<td>INTEGER</td>
<td>Linux process identifier of the client process that issued this connection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The client process could be on a different machine from the server.</td>
</tr>
<tr>
<td>CLIENT_LABEL</td>
<td>VARCHAR</td>
<td>User-specified label for the client connection that can be set when using ODBC. See Label in DSN Parameters in Connecting to Vertica.</td>
</tr>
<tr>
<td>SSL_STATE</td>
<td>VARCHAR</td>
<td>Indicates if Vertica used Secure Socket Layer (SSL) for a particular session. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- None – Vertica did not use SSL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Server – Sever authentication was used, so the client could authenticate the server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mutual – Both the server and the client authenticated one another through mutual authentication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>See Implementing Security and TLS/SSL Server Authentication</strong> in the Administrator's Guide.</td>
</tr>
<tr>
<td>AUTHENTICATION_METHOD</td>
<td>VARCHAR</td>
<td>Type of client authentication used for a particular session, if known. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Trust</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reject</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Kerberos</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MD5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LDAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Kerberos-GSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ident</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Security and Authentication and Implementing Client Authentication.</td>
</tr>
<tr>
<td>CLIENT_TYPE</td>
<td>VARCHAR</td>
<td>The type of client from which the connection was made. Possible client type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ADO.NET Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ODBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• JDBC Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• vsql</td>
</tr>
<tr>
<td>CLIENT_VERSION</td>
<td>VARCHAR</td>
<td>Returns the client version.</td>
</tr>
<tr>
<td>CLIENT_OS</td>
<td>VARCHAR</td>
<td>Returns the client operating system.</td>
</tr>
<tr>
<td>CLIENT_OS_USER_NAME</td>
<td>VARCHAR</td>
<td>The name of the user that logged into, or attempted to log into, the database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is logged even when the login attempt is unsuccessful.</td>
</tr>
</tbody>
</table>

**Privileges**

No explicit privileges are required. You only see the records for tables that you have privileges to view.
See Also

- CURRENT_SESSION
- SESSION_PROFILES
- SESSIONS
- SYSTEM_SESSIONS

WOS_CONTAINER_STORAGE

Monitors information about WOS storage, which is divided into regions. Each region allocates blocks of a specific size to store rows.

Note: The WOS allocator can use large amounts of virtual memory without assigning physical memory.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_NAME</td>
<td>VARCHAR</td>
<td>The node name for which information is listed.</td>
</tr>
<tr>
<td>WOS_TYPE</td>
<td>VARCHAR</td>
<td>Returns one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- system – for system table queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- user - for other user queries</td>
</tr>
<tr>
<td>WOS_ALLOCATION_REGION</td>
<td>VARCHAR</td>
<td>The block size allocated by region in KB. The summary line sums the amount of memory used by all regions.</td>
</tr>
<tr>
<td>REGION_VIRTUAL_SIZE_KB</td>
<td>INTEGER</td>
<td>The amount of virtual memory in use by region in KB. Virtual size is greater than or equal to allocated size, which is greater than or equal to in-use size.</td>
</tr>
<tr>
<td>REGION_ALLOCATED_SIZE_KB</td>
<td>INTEGER</td>
<td>The amount of physical memory in use by a particular region in KB.</td>
</tr>
<tr>
<td>REGION_IN_USE_SIZE_KB</td>
<td>INTEGER</td>
<td>The actual amount of data stored by the region in</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KB</td>
<td></td>
<td>KB. The amount of memory used by the WOS is typically capped at one quarter physical memory per node.</td>
</tr>
<tr>
<td>REGION_SMALL_RELEASE_COUNT</td>
<td>INTEGER</td>
<td>Internal use only</td>
</tr>
<tr>
<td>REGION_BIG_RELEASE_COUNT</td>
<td>INTEGER</td>
<td>Internal use only</td>
</tr>
<tr>
<td>EXTRA_RESERVED_BYTES</td>
<td>INTEGER</td>
<td>The amount of extra memory allocated to maintain WOS sort information.</td>
</tr>
<tr>
<td>EXTRA_USED_BYTES</td>
<td>INTEGER</td>
<td>The amount of memory in use currently to maintain the WOS sort information.</td>
</tr>
</tbody>
</table>
Appendix: Compatibility with Other RDBMS

This section describes compatibility of Vertica with other relational database management systems.

Information in this appendix is intended to simplify database migration to Vertica.

Data Type Mappings Between Vertica and Oracle

Oracle uses proprietary data types for all main data types (for example, VARCHAR, INTEGER, FLOAT, DATE), if you plan to migrate your database from Oracle to Vertica, [[[Undefined variable _Branding_Variables._Company_Acronym]]] strongly recommends that you convert the schema—a simple and important exercise that can minimize errors and time lost spent fixing erroneous data issues.

The following table compares the behavior of Oracle data types to Vertica data types.

<table>
<thead>
<tr>
<th>Oracle</th>
<th>Vertica</th>
<th>Notes</th>
</tr>
</thead>
</table>
| NUMBER (no explicit precision)  | INTEGER, NUMERIC or FLOAT | In Oracle, the NUMBER data type with no explicit precision stores each number N as an integer M, together with a scale S. The scale can range from -84 to 127, while the precision of M is limited to 38 digits. So \( N = M \times 10^S \).

When precision is specified, precision/scale applies to all entries in the column. If omitted, the scale defaults to 0.

For the common case where Oracle’s NUMBER with no explicit precision data type is used to store only integer values, INTEGER is the best suited and the fastest Vertica data type. However, INTEGER (the same as BIGINT) is limited to a little less than 19 digits, with a scale of 0; if the Oracle column contains integer values outside of the range \([-9223372036854775807, +9223372036854775807]\), use the Vertica data type NUMERIC\((p,0)\) where \(p\) is the maximum number of digits required to represent the
<table>
<thead>
<tr>
<th>Oracle</th>
<th>Vertica</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{NUMBER (P,0),} \hspace{1cm} P \leq 18</td>
<td>\texttt{INTEGER}</td>
<td>In Oracle, when precision is specified the precision/scale applies to all entries in the column. If omitted the scale defaults to 0. For the Oracle \texttt{NUMBER} data type with 0 scale, and a precision less than or equal to 18, use \texttt{INTEGER} in Vertica.</td>
</tr>
<tr>
<td>\texttt{NUMBER (P,0),} \hspace{1cm} P &gt; 18</td>
<td>\texttt{NUMERIC (p,0)}</td>
<td>An Oracle column precision greater than 18 is often more than an application really needs. If all values in the Oracle column are within the INT range ([-9223372036854775807,+9223372036854775807]), use \texttt{INTEGER} for best performance. Otherwise, use the Vertica data type \texttt{NUMERIC(p,0)}, where \texttt{p = P}.</td>
</tr>
<tr>
<td>\texttt{NUMBER (P,S)} \hspace{1cm} all cases other than previous</td>
<td>\texttt{NUMERIC (p,s)} or \texttt{FLOAT}</td>
<td>When \texttt{P} \geq \texttt{S} and \texttt{S} \geq 0, use \texttt{p = P} and \texttt{s = S}, unless the data allows reducing \texttt{P} or using \texttt{FLOAT} as discussed above. If \texttt{S} &gt; \texttt{P}, use \texttt{p = S}, \texttt{s = S}. If \texttt{S} &lt; 0, use \texttt{p = P - S}, \texttt{s = 0}.</td>
</tr>
</tbody>
</table>

values of \textit{N}.

Even though no explicit scale is specified for an Oracle \texttt{NUMBER} column, Oracle allows non-integer values, each with its own scale. If the data stored in the column is approximate, Vertica recommends using the Vertica data type \texttt{FLOAT}, which is standard IEEE floating point, like \texttt{ORACLE BINARY\_DOUBLE}. If the data is exact with fractional places, for example dollar amounts, Vertica recommends \texttt{NUMERIC(p,s)} where \texttt{p} is the precision (total number of digits) and \texttt{s} is the maximum scale (number of decimal places).

Vertica conforms to standard SQL, which requires that \texttt{p} \geq \texttt{s} and \texttt{s} \geq 0. Vertica’s \texttt{NUMERIC} data type is most effective for \texttt{p=18}, and increasingly expensive for \texttt{p=37, 58, 67, etc.}, where \texttt{p} \leq 1024.

Vertica recommends against using the data type \texttt{NUMERIC (38,s)} as a default "failsafe" mapping to guarantee no loss of precision. \texttt{NUMERIC (18,s)} is better, and \texttt{INTEGER} or \texttt{FLOAT} are better yet, if one of these data types will do the job.
<table>
<thead>
<tr>
<th>Oracle</th>
<th>Vertica</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC (P,S)</td>
<td>See notes --&gt;</td>
<td>Rarely used in Oracle. See notes for the NUMBER type.</td>
</tr>
<tr>
<td>DECIMAL (P,S)</td>
<td>See notes --&gt;</td>
<td>DECIMAL is a synonym for NUMERIC. See notes for the NUMBER type.</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>FLOAT</td>
<td>Same as FLOAT(53) or DOUBLE PRECISION.</td>
</tr>
<tr>
<td>BINARYDOUBLE</td>
<td>FLOAT</td>
<td>Same as FLOAT(53) or DOUBLE PRECISION.</td>
</tr>
<tr>
<td>RAW</td>
<td>VARBINARY (RAW)</td>
<td>The maximum size of RAW in Oracle is 2,000 bytes. The maximum size of CHAR/BINARY in Vertica is 65000 bytes. In Vertica, RAW is a synonym for VARBINARY.</td>
</tr>
<tr>
<td>LONG RAW</td>
<td>VARBINARY (RAW)</td>
<td>The maximum size of Oracle’s LONG RAW is 2GB. The maximum size of Vertica’s VARBINARY is 65000 bytes. Vertica users should exercise caution to avoid truncation during data migration from Oracle.</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>CHAR(n)</td>
<td>The maximum size of CHAR in Oracle is 2,000 bytes. The maximum size of CHAR in Vertica is 65000 bytes.</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>CHAR(n*3)</td>
<td>Vertica supports national characters with CHAR(n) as variable-length UTF8-encoded UNICODE character string. UTF-8 represents ASCII in 1 byte, most European characters in 2 bytes, and most oriental and Middle Eastern characters in 3 bytes.</td>
</tr>
<tr>
<td>VARCHAR2(n)</td>
<td>VARCHAR(n)</td>
<td>The maximum size of VARCHAR2 in Oracle is 4,000 bytes. The maximum size of VARCHAR in Vertica is 65000.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The behavior of Oracle’s VARCHAR2 and Vertica’s VARCHAR is semantically different. Vertica’s VARCHAR exhibits standard SQL behavior, whereas Oracle’s VARCHAR2 is not completely consistent with standard behavior – it treats an empty string as NULL value and uses</td>
</tr>
<tr>
<td>Oracle</td>
<td>Vertica</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NVARCHAR2 (n)</td>
<td>VARCHAR (n*3)</td>
<td>non-padded comparison if one operand is VARCHAR2.</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP or possibly DATE</td>
<td>See notes for NCHAR().</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle’s DATE is different from the SQL standard DATE data type implemented by Vertica. Oracle’s DATE includes the time (no fractional seconds), while Vertica DATE type includes only date per SQL specification.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>TIMESTAMP defaults to six places, that is, to microseconds</td>
</tr>
<tr>
<td>TIMESTAMP WITH</td>
<td>TIMESTAMP WITH</td>
<td>TIME ZONE defaults to the currently SET or system time zone.</td>
</tr>
<tr>
<td>TIME ZONE</td>
<td>TIME ZONE</td>
<td>Per the SQL standard, INTERVAL can be qualified with YEAR TO MONTH sub-type in Vertica.</td>
</tr>
<tr>
<td>INTERVAL YEAR</td>
<td>INTERVAL YEAR</td>
<td>In Vertica, DAY TO SECOND is the default sub-type for INTERVAL.</td>
</tr>
<tr>
<td>TO MONTH</td>
<td>TO MONTH</td>
<td></td>
</tr>
<tr>
<td>INTERVAL DAY</td>
<td>INTERVAL DAY</td>
<td>You can store a CLOB (character large object) or BLOB (binary large object) value in a table or in an external location. The maximum size of a CLOB or BLOB is 128 TB. The maximum size of the LONG data types is 32,000,000 bytes.</td>
</tr>
<tr>
<td>TO SECOND</td>
<td>TO SECOND</td>
<td></td>
</tr>
<tr>
<td>CLOB, BLOB</td>
<td>LONG VARCHAR, LONG VARBINARY</td>
<td>Oracle recommends using CLOB and BLOB data types instead of LONG and LONG RAW data types.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In Oracle, a table can contain only one LONG column. The maximum size of a LONG or LONG RAW data type is 2 GB.</td>
</tr>
</tbody>
</table>
Vertica provides tools and features that allow you to ensure your system is secure as well as to prevent unauthorized users from accessing sensitive information.

Client Authentication establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server.
Client Authentication

Implementing strong security programs provides Vertica users the assurance that access to sensitive information is closely guarded. Vertica uses several approaches to manage data access.

The database server uses client authentication to establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server using the supplied credentials.

Encrypting Client-Server Communication

Vertica uses Secure Socket Layer (SSL) and Transport Layer Security (TLS) to establish a secure connection between the client machine and the server. Configure SSL/TLS to:

- Authenticate the server so the client can confirm the server's identity. Vertica supports mutual authentication in which the server can also confirm the identity of the client. This authentication helps prevent "man-in-the-middle" attacks.

- Encrypt data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised.

- Verify that data sent between the client and server has not been altered during transmission.

For details see TLS/SSL Server Authentication.

Authentication Management

Users with the DBADMIN Role can manage the following authentication tasks:

- Create authentication records using CREATE AUTHENTICATION.
Important: Configure client authentication so that the DBADMIN user can always access the database locally. If a problem occurs with the authentication that blocks all users from logging in, the DBADMIN user needs access to correct the problem.

- Assign a specific authentication method to a user using GRANT (Authentication).
- Use ALTER AUTHENTICATION to:
  - Enable/disable authentication methods.
  - Define a default authentication method to be used if a user has not been assigned a specific authentication method.
- Define parameters required by LDAP, Ident, and Kerberos authentication methods.
- Revoke a user's authentication record using REVOKE Authentication. This user now uses the default authentication.
- Delete an authentication record from the database using DROP AUTHENTICATION. Any users assigned the dropped record now use the default authentication method.

For details about managing authentication records, see:

- DBADMIN Authentication Access
- Creating Authentication Records
- Enabling and Disabling Authentication Methods
- Granting and Revoking Authentication Methods
- Modifying Authentication Records

See Implementing Client Authentication.

User Authorization

Database users should have access to just the database resources they need to perform their required tasks. For example, some users need to query only specific sets of data. To prevent unauthorized access to additional data, you can limit their access to just the data that they need to run their queries. Other users should be able to read the data but not be able to modify or insert new data. Still other users might need more permissive access, including the
right to create and modify schemas, tables, and views, or grant other users access to database resources.

For information on controlling data access, see the following:

- About Database Users in Managing Users and Privileges
- About Database Roles to grant users access to a set of privileges.
- Access Policies to limit user's from viewing data from a specific table.

**Implementing Client Authentication**

Vertica restricts which database users can connect through client authentication. The database server uses client authentication to establish the identity of the requesting client and determines whether that client is authorized to connect to the Vertica server using the supplied credentials.

When a user or client application connect to the Vertica database server, it supplies a unique user name and password to gain access.

Vertica offers several client authentication methods. You can configure Vertica to require just a user name for connections, you likely require more secure means of authentication, such as a password at a minimum.

**Note:** Topics in this section describe authentication methods supported at the database server layer. For information on authentication between server and client, see TLS/SSL Server Authentication.

**How Client Authentication Works**

When connecting to a Vertica database, a user or client application must supply the name of a valid user account. In addition, the application usually includes a means of authentication, such as a password or security certificate.

There are two types of client authentication:

- LOCAL—Authenticating users or applications that are trying to connect from the same node that the database is running on.
HOST—Authenticating users or applications that are trying to connect from a node that has a different IPv4 or IPv6 address than the database.

The DBADMIN user manages the client authentication information that the database uses to authenticate users.

Vertica takes the following steps to authenticate users:

1. When a user or application attempts to connect to a Vertica database, the system checks to see if the user is a DBADMIN user. If so, authentication occurs using the assigned authentication method, local trust or local hash authentication.

2. For non-DBADMIN users, the database checks to see if the user is associated with an authentication method through a GRANT statement. If so, the database allows the user to log in if they match the parameters required for that authentication method.

   Note: For detailed information on how authentication priorities work, see Priorities for Client Authentication Methods.

The DBADMIN user can grant an authentication method to users or user roles. The DBADMIN user can also create a default authentication method that Vertica uses when no authentication has been associated with a user or role.

3. If the user has not been granted an authentication method, the database checks to see if the DBADMIN has established a default authentication method.

4. If the DBADMIN has specified a default authentication method, the database authenticates the user using that default method.

5. If you have not specified a default authentication method, the database checks to see if the DBADMIN user has defined any authentication methods. If not, no authentication information exists in the database. However, if a password exists, the DBADMIN user can log in.

6. If authentication information exists, Vertica rejects the user request to connect to the database. The DBADMIN has not granted an authentication method for that user nor has the DBADMIN defined a default authentication method for all users ('public').

7. If no authentication records exist in the database, Vertica uses implicit trust/implicit password to authenticate the user.

The following image illustrates the steps involved in client authentication:
IPv4 and IPv6 for Client Authentication

Vertica supports clients using either the IPv4 or the IPv6 protocol to connect to the database server. Internal communication between database servers must consistently use one address.
family (IPv4 or IPv6). The client, however, can connect to the database from either type of IP address.

If the client will be connecting from either IPv4 or IPv6, you must create two authentication methods, one for each address. Any authentication method that uses HOST authentication requires an IP address.

For example, the first statement allows users to connect from any IPv4 address. The second statement allows users to connect from any IPv6 address:

```sql
=> CREATE AUTHENTICATION <name> METHOD 'gss' HOST '0.0.0.0/0'; --IPv4
=> CREATE AUTHENTICATION <name> METHOD 'gss' HOST '::/0'; --IPv6
```

If you are using a literal IPv6 address in a URL, you must enclose the IPv6 address in square brackets as shown in the following examples:

```sql
=> ALTER AUTHENTICATION Ldap SET host='ldap://[1dfa:2bfa:3:45:5:6:7:877]';
=> ALTER AUTHENTICATION Ldap SET host='ldap://[fdfb:dbfa:0:65::177]';
=> ALTER AUTHENTICATION Ldap SET host='ldap://[fdfb::177]';
=> ALTER AUTHENTICATION Ldap SET host='ldap://[:1]';
=> ALTER AUTHENTICATION Ldap SET host='ldap://[1dfa:2bfa:3:45:5:6:7:877]:5678';
```

If you are working with a multi-node cluster, any IP/netmask settings in (HOST, HOST TLS, HOST NO TLS) must match all nodes in the cluster. This setup allows the database owner to authenticate with and administer every node in the cluster. For example, specifying 10.10.0.8/30 allows a CIDR address range of 10.10.0.8–10.10.0.11.

For detailed information about IPv6 addresses, see RFC 1924 and RFC 2732.

**Supported Client Authentication Methods**

Vertica supports the following types of authentication to prove a client's identity.

- **Trust**—Authorizes any user that connects to the database using a valid user name, no password is required and authentication is not performed.

- **Reject**—Rejects the connection attempt when a user with an invalid user name attempts to connect to the database.

- **Kerberos (GSS)**—Authorizes connecting to the database using a MIT Kerberos implementation. The KDC must support Kerberos 5 using GSS-API. This API also provides compatibility with non-MIT Kerberos implementations, such as Java and Windows clients.
- Hash—Sends encrypted passwords hashed by the MD5 algorithm or the more secure SHA-512 method over the network. The server provides the client with salt.

- LDAP—Works like password authentication except the LDAP method authenticates the client against a Lightweight Directory Access Protocol or Active Directory server.

- Ident—Authenticates the client against the username in an Ident server.

- TLS authentication—Authenticates the client using digital certificates that contain a public key. Transport Layer Security (TLS) is the successor to Secure Sockets Layer (SSL) authentication.

## Local and Host Authentication

You can define a client authentication method as:

- Local: Local connection to the database.

- Host: Remote connection to the database from different hosts, each with their own IPv4 or IPv6 address and host parameters. For more information see IPv4 and IPv6 for Client Authentication above.

Some authentication methods cannot be designated as local, as listed in this table:

<table>
<thead>
<tr>
<th>Authentication Method</th>
<th>Local?</th>
<th>Host?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kerberos (GSS)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ident</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LDAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Hash</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reject</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Trust</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Authentication for Chained Users and Roles

Vertica supports creating chained users and roles, where you can grant ROLE2 privileges to ROLE1. All users in ROLE1 use the same authentication assigned to ROLE2. For example:

```sql
=> CREATE USER user1;
=> CREATE ROLE role1;
=> CREATE ROLE role2;
=> CREATE AUTHENTICATION h1 method 'hash' local;
=> GRANT AUTHENTICATION h1 to role2;
=> GRANT role2 to role1;
=> GRANT role1 to user1;
```

The user and role chain in the example above can be illustrated as follows:

auth1 -> role2 -> role1 -> user1

In this example, since role2 privileges are granted to role1 you only need to grant authentication to role2 to also enable it for role1.

DBADMIN Authentication Access

The DBADMIN user must have access to the database at all times.

The DBADMIN account must authenticate against the database using local trust or local hash authentication.

Vertica recommends that you create an authentication method (LOCAL TRUST or LOCAL PASSWORD) with a very high priority, say, 10,000. Grant this method to the DBADMIN user and set the priority using ALTER AUTHENTICATION.

With the high priority, this new authentication method supersedes any authentication methods you create for PUBLIC (which includes the DBADMIN user). Even if you make changes to PUBLIC authentication methods, the DBADMIN user can now connect to the database at any time.

This example shows how you configure local trust authentication for the DBADMIN user. As a result, the user can use vsql with the -h option and does not need to enter a password:

```sql
=> CREATE AUTHENTICATION v_dbadmin_trust METHOD 'trust' LOCAL;
=> GRANT AUTHENTICATION v_dbadmin_trust TO dbadmin;
=> ALTER AUTHENTICATION v_dbadmin_trust PRIORITY 10000;
```
The next example shows how you configure host hash authentication for DBADMIN. They allow the user to access the Vertica database using the assigned password from any IPv4 address. The DBADMIN user can access the database using vsq1 -h --host, the Administration Tools, or any other tools that connects to Vertica:

```sql
=> CREATE AUTHENTICATION v_dbadmin_hash METHOD 'hash' HOST '0.0.0.0/0';
=> GRANT AUTHENTICATION v_dbadmin_hash TO dbadmin;
=> ALTER AUTHENTICATION v_dbadmin_hash PRIORITY 10000;
=> SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'SHA512');
```

Note: Vertica supports IPv4 and IPv6 addresses. For more information, see IPv4 and IPv6 for Client Authentication.

## Creating Authentication Records

You can manage client authentication records using vsq1 commands. To use these statements, you must be connected to the database.

**Important:** You cannot modify client authentication records using the Administration Tools. The Administration Tools interface allows you to modify the contents of the `vertica.conf` file. However, Vertica ignores any client authentication information stored in that file.

You create authentication records with `CREATE AUTHENTICATION`, which Vertica stores in the catalog and automatically enables.

### Examples

The following examples show how to create authentication records.

Create authentication method `localpwd` to authenticate users who are trying to log in from a local host using a password:

```sql
=> CREATE AUTHENTICATION localpwd METHOD 'hash' LOCAL;
```

Create authentication method `v_ldap` that uses LDAP over TLS to authenticate users logging in from the host with the IPv4 address 10.0.0.0/23:

```sql
=> CREATE AUTHENTICATION v_ldap METHOD 'ldap' HOST TLS '10.0.0.0/23';
```

Create authentication method `v_kerberos` to authenticate users who are trying to connect from any host in the networks 2001:0db8:0001:12xx:
CREATE AUTHENTICATION v_kerberos METHOD 'gss' HOST '2001:db8:1::1200/56';

Create authentication method RejectNoSSL that rejects users from any IP address that are trying to authenticate without SSL/TLS:

CREATE AUTHENTICATION RejectNoSSL_IPv4 METHOD 'reject' HOST NO TLS '0.0.0.0/0'; --IPv4
CREATE AUTHENTICATION RejectNoSSL_IPv6 METHOD 'reject' HOST NO TLS '::/0'; --IPv6

See Also

- Deleting Authentication Records
- Enabling and Disabling Authentication Methods
- Granting and Revoking Authentication Methods
- Modifying Authentication Records

Modifying Authentication Records

To modify existing authentication records, you must first be connected to your database. The following examples show how to make changes to your authentication records. For more information see ALTER AUTHENTICATION.

Rename an Authentication Method

Rename the v_kerberos authentication method to K5, and enable it. All users who have been associated with the v_kerberos authentication method are now associated with the K5 method granted instead.

ALTER AUTHENTICATION v_kerberos RENAME TO K5 ENABLE;
Specify a Priority for an Authentication Method

Specify a priority of 10 for K5 authentication:

```sql
=> ALTER AUTHENTICATION K5 PRIORITY 10;
```

For more information see Priorities for Client Authentication Methods.

Change a Parameter

Set the `system_users` parameter for `ident1` authentication to `root`:

```sql
=> CREATE AUTHENTICATION ident1 METHOD 'ident' LOCAL;
=> ALTER AUTHENTICATION ident1 SET system_users='root';
```

Change the IP address and specify the parameters for an LDAP authentication method named Ldap1.

In this example, you specify the bind parameters for the LDAP server. Vertica connects to the LDAP server, which authenticates the Vertica client. If the authentication succeeds, Vertica authenticates any users who have been granted the Ldap1 authentication method on the designated LDAP server:

```sql
=> CREATE AUTHENTICATION Ldap1 METHOD 'ldap' HOST '172.16.65.196';
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',
    binddn_prefix='cn=', binddn_suffix=',dc=qa_domain,dc=com';
```

Change the IP address, and specify the parameters for an LDAP authentication method named Ldap1. Assume that Vertica does not have enough information to create the distinguished name (DN) for a user attempting to authenticate. Therefore, in this case, you must specify to use LDAP search and bind:

```sql
=> CREATE AUTHENTICATION Ldap1 METHOD 'ldap' HOST '172.16.65.196';
=> ALTER AUTHENTICATION Ldap1 SET host='ldap://172.16.65.177',
    basedn='dc=qa_domain,dc=com', binddn='cn=Manager,dc=qa_domain,dc=com',
    search_attribute='cn', bind_password='secret';
```
Change the Associated Method

Change the localpwd authentication from trust to hash:

```
=> CREATE AUTHENTICATION localpwd METHOD 'trust' LOCAL;
=> ALTER AUTHENTICATION localpwd METHOD 'hash';
```

ALTER AUTHENTICATION validates the parameters you enter. If there are errors, it disables the authentication method that you are trying to modify.

Using the Administration Tools

The advantages of using the Administration Tools are:

- You do not have to connect to the database
- The editor verifies that records are correctly formed
- The editor maintains records so they are available to you to edit later

**Note:** You must restart the database to implement your changes.

For information about using the Administration Tools to create and edit authentication records, see [Creating Authentication Records](#).

Using the Client Authentication Configuration Parameter

The advantage of using the ClientAuthentication configuration parameter is that the changes are implemented immediately across all nodes within the database cluster. You do not need to restart the database.

However, all the database nodes must be up and you must [connect to the database](#) before you set this parameter. Most importantly, this method does not verify that records are correctly formed and it does not maintain the records so you can modify them later.
New authentication records are appended to the list of existing authentication records. Because Vertica scans the list of records from top to bottom and uses the first record that matches the incoming connection, you might find your newly-added record does not have an effect if Vertica used an earlier record instead.

To configure client authentication through a connection parameter, use the `ALTER DATABASE` statement:

```
=> ALTER DATABASE exampledb SET ClientAuthentication = 'connection type user name address method';
```

When you specify authentication records, make sure to adhere to the following guidelines:

- Fields that make up the record can be separated by white space or tabs
- Other than IP addresses and mask columns, field values cannot contain white space

### Examples

The following example creates an authentication record for the trust method:

```
=> ALTER DATABASE exampledb SET ClientAuthentication = 'hostssl dbadmin 0.0.0.0/0 trust';
```

The following example creates an authentication record for the LDAP method:

```
=> ALTER DATABASE exampledb SET ClientAuthentication = 'host all 10.0.0.0/8
ldap "ldap://summit.vertica.com;cn=;,dc=vertica,dc=com"';
```

The following example specifies three authentication records. In a single command, separate each authentication record by a comma:

```
=> ALTER DATABASE exampledb SET ClientAuthentication = 'hostssl dbadmin 0.0.0.0/0 trust, hostssl all 0.0.0.0/0 md5, local all trust';
```

### Deleting Authentication Records

To delete client authentication record, use `DROP AUTHENTICATION`. To use this approach, you have to be connected to your database.

To delete an authentication record for md5_auth use the following command:

```
=> DROP AUTHENTICATION md5_auth;
```

To delete an authentication record for a method that has been granted to a user, use the `CASCADE` keyword:
CREATE AUTHENTICATION localpwd METHOD 'password' LOCAL;
GRANT AUTHENTICATION localpwd TO jsmith;
DROP AUTHENTICATION localpwd CASCADE;

See Also

- Creating Authentication Records
- Granting and Revoking Authentication Methods

Priorities for Client Authentication Methods

You can associate one or more authentication methods to a connection or user. For a user who has multiple authentication methods, specify the order in which Vertica should try them. To do so, assign a priority to each authentication method using ALTER AUTHENTICATION. All priority values should be a non-negative INTEGER.

Higher values indicate higher priorities. Vertica tries to authenticate a user with an authentication method in order of priority from highest to lowest. For example:

- A priority of 10 is higher than a priority of 5.
- A priority 0 is the lowest possible value.

Important: Vertica does not support authentication chaining where you can configure multiple authentication modules to identify a specific user. For example, chaining to try a password authentication method after an LDAP authentication method failed is not supported.

Priority Order for Authentication Methods

When you associate multiple authentication methods with a connection, Vertica uses the following order to determine how to authenticate the client:

- Administrator-assigned priority for an individual method
- The most specific IP addresses have priority over the least specific IP addresses
For example, the IPv4 address 10.3.4.128/25 has priority over 10.3.0.0/24, which in turn has priority over 10.3.0.0/16. The IPv6 address 2001:db8:ab::123/128 has priority over 2001:db8:1::1200/56.

- Reject
- GSS | LDAP | Ident
- Hash
- Trust

**Authentication Attempts Using Multiple Methods**

If there is only one authentication method associated with a user, Vertica uses that method to authenticate the login attempt.

If the administrator has associated multiple authentication methods with a given user or IP address, Vertica tries to authenticate as follows:

- If the highest priority authentication method is Ident and authentication fails, Vertica tries the next highest priority authentication method, regardless of what method it uses.

  If the next attempt does not use Ident authentication and fails, the authentication process ends. However, if the next attempt uses Ident and fails, Vertica continues to the next highest priority method. This process continues until authentication is successful or a non-Ident authentication attempt fails.

- If the highest priority method is LDAP and authentication fails, Vertica searches for the next highest priority LDAP method. Authentication attempts continue until the authentication is successful, or there are no additional LDAP authentication methods that satisfy the connection criteria.

  Note that if a user not found error occurs during LDAP authentication, the retry connection attempt initiates only if you set the ldap_continue parameter to yes.

- For all other authentication types, Vertica tries the highest priority authentication method associated with that user. If that authentication fails, the authentication process stops.

For example, suppose there are two client authentication methods associated with a user, as follows:
When user tries to connect to the database, Vertica first tries auth_name2 to authenticate because it has a higher priority. If that fails, Vertica tries auth_name1. If that fails, authentication fails.

### Specifying Authentication Method Priority

To specify priorities for client authentication methods, use `ALTER AUTHENTICATION`. The priority value must be a non-negative INTEGER. Higher numbers indicate a higher priority. The default value, 0, is the lowest possible priority.

The syntax is:

```
ALTER AUTHENTICATION <name> ... PRIORITY <priority_value>;
```

If you do not specify a priority, or omit the `<priority_value>` when using `ALTER AUTHENTICATION`, Vertica sets the priority to 0.

### DBADMIN and Authentication Priority

To allow the DBADMIN user to connect to the database at any time, Vertica recommends that you create an authentication method (LOCAL TRUST or LOCAL PASSWORD) with a very high priority, such as 10,000. Grant this method to the DBADMIN user, and set the priority using `ALTER AUTHENTICATION`.

With the high priority, this new authentication method supersedes any authentication methods you create for PUBLIC (which includes the DBADMIN user). Even if you make changes to PUBLIC authentication methods, the DBADMIN still has access.

**Note:** For the DBADMIN user to be able to perform all Admintools functions, the DBADMIN must always be able to authenticate by LOCAL TRUST or LOCAL PASSWORD (the default for DBADMIN user). If you have changed DBADMIN user authentication from LOCAL TRUST or LOCAL PASSWORD, use the `ALTER AUTHENTICATION` statement to once again give the DBADMIN user LOCAL TRUST or LOCAL PASSWORD authentication.
Viewing Information About Client Authentication Records

For information about client authentication records that you have configured for your database, query the following system tables in the V_CATALOG schema:

- CLIENT_AUTH
- CLIENT_AUTH_PARAMS
- PASSWORD_AUDITOR
- USER_CLIENT_AUTH

To determine the details behind the client authentication used for a particular user session, query the following tables in the V_MONITOR schema:

- SESSIONS
- USER_SESSIONS

Enabling and Disabling Authentication Methods

When you create an authentication method, Vertica stores it in the catalog and enables it automatically. To enable or disable an authentication method, use the ALTER AUTHENTICATION statement. To use this approach, you must be connected to your database.

If an authentication method has not been enabled, Vertica cannot use it to authenticate users and clients trying to connect to the database.

To enable an authentication method:

```
ALTER AUTHENTICATION v_kerberos ENABLE;
```

To disable this authentication method:

```
ALTER AUTHENTICATION v_kerberos DISABLE;
```
See Also

- Creating Authentication Records
- Deleting Authentication Records
- Granting and Revoking Authentication Methods
- Modifying Authentication Records

Granting and Revoking Authentication Methods

Before Vertica can validate a user or client through an authentication method, you must first associate that authentication method with the user or role that requires it. To do this, use GRANT AUTHENTICATION. When that user or role no longer needs to connect to Vertica using that method, you can disassociate that authentication from that user with REVOKE AUTHENTICATION.

Grant Authentication Methods

You can grant an authentication method to a specific user or role. You can also specify the default authentication method by granting an authentication method to Public. Use the GRANT (Authentication) statement as follows:

This example uses a GRANT AUTHENTICATION statement to associate v_ldap authentication with user jsmith:

```sql
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```

This example uses a GRANT AUTHENTICATION statement to associate v_gss authentication to the role DBprogrammer:

```sql
=> CREATE ROLE DBprogrammer;
=> GRANT AUTHENTICATION v_gss to DBprogrammer;
```

This example sets the default client authentication method to v_localpwd:
=> GRANT AUTHENTICATION v_localpwd TO Public;

Revoke Authentication Methods

If you no longer want to authenticate a user or client with a given authentication method, use the `REVOKE (Authentication)` statement as follows:

This example revokes `v_ldap` authentication from user `jsmith`:

=> REVOKE AUTHENTICATION v_ldap FROM jsmith;

This example revokes `v_gss` authentication from the role `DBprogrammer`:

=> REVOKE AUTHENTICATION v_gss FROM DBprogrammer;

This example removes `localpwd` as the default client authentication method:

=> REVOKE AUTHENTICATION localpwd from Public;

Hash Authentication

Vertica provides a hash authentication method that allows you to use the MD5 algorithm or the more secure algorithm, SHA-512, to store user passwords. SHA-512 is one of the industry-standard SHA-2 family of hash algorithms that address weaknesses in SHA-1 and MD5. For information on creating passwords to be hashed during authentication see Passwords

Note: Vertica strongly recommends that you use SHA-512 for hash authentication because it is more secure than MD5.

Before you perform hash authentication, review the following topics:

- **Hash Authentication Parameters**—Describes the two hash authentication parameters that specify which hashing algorithm to use.

- **Upgrade Considerations for Hash Authentication**—Before you implement the SHA-512 algorithm for one or more users, you must be aware of several issues. For details, review this topic before proceeding.

- **How to Configure Hash Authentication**—After you decide to implement hash authentication in your database, follow the steps described in this topic.
Hash Authentication Parameters

Two parameters control which algorithm hash authentication uses for hashing and storing user passwords:

- A user-level parameter, Security_Algorithm:

  => ALTER USER username SECURITY_ALGORITHM 'MD5' IDENTIFIED BY 'newpassword';
  => ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';

- A system-level configuration parameter, SecurityAlgorithm:

  => SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'MD5');
  => SELECT SET_CONFIG_PARAMETER('SecurityAlgorithm', 'SHA512');

Both parameters can have the following values:

- 'NONE'
- 'MD5'
- 'SHA512'

Note: If your current password is in the MD5 format you cannot rename a user with ALTER USER.

The user-level parameter usually has precedence over the system-level parameter. However, if the user-level parameter is 'NONE', Vertica hashes passwords with the algorithm assigned to the system-level parameter value. If both parameters are 'NONE', Vertica uses the MD5 algorithm.

These values, which are stored in the PASSWORD_AUDITOR system table, affect the security algorithm that is actually used for hash authentication.

<table>
<thead>
<tr>
<th>User-Level Parameter Value</th>
<th>System-Level Parameter Value</th>
<th>Algorithm Used for Hash Authentication</th>
<th>Algorithm Used for Hash Authentication - FIPS mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NONE'</td>
<td>'NONE'</td>
<td>MD5</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'NONE'</td>
<td>'MD5'</td>
<td>MD5</td>
<td>SHA-512</td>
</tr>
<tr>
<td>User-Level Parameter Value</td>
<td>System-Level Parameter Value</td>
<td>Algorithm Used for Hash Authentication</td>
<td>Algorithm Used for Hash Authentication - FIPS mode</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>----------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>'NONE'</td>
<td>'SHA512'</td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'MD5'</td>
<td>'NONE'</td>
<td>MD5</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'MD5'</td>
<td>'MD5'</td>
<td>MD5</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'MD5'</td>
<td>'SHA512'</td>
<td>MD5</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'SHA512'</td>
<td>'NONE'</td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'SHA512'</td>
<td>'MD5'</td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
<tr>
<td>'SHA512'</td>
<td>'SHA512'</td>
<td>SHA-512</td>
<td>SHA-512</td>
</tr>
</tbody>
</table>

How to Configure Hash Authentication

Follow these steps to configure hash authentication:

1. Create an authentication method that is based on hash encryption. When you create an authentication method, it is automatically enabled for use.

   The following example shows how to create an authentication method v_hash for users logging in from the IP address 10.0.0.0/0.

   ```
   => CREATE AUTHENTICATION v_hash METHOD 'hash' HOST '10.0.0.0/0';
   ```

   If users are trying to connect from an IPv6 address, the statement might look like this example:

   ```
   => CREATE AUTHENTICATION v_hash METHOD 'hash' HOST '2001:db8:ab::123/128';
   ```

2. Decide which password-hashing algorithm you want to use: MD5 or the more secure SHA-512.

3. Specify the security algorithm as follows:
At the system level, set the SecurityAlgorithm configuration parameter. This setting applies to all users, unless their user-level security is set to another value:

```sql
=> ALTER DATABASE mydb SET SecurityAlgorithm = 'MD5';
=> ALTER DATABASE mydb SET SecurityAlgorithm = 'SHA512';
```

If you want users to inherit the system-level security, set their passwords to expire immediately. Users must change their passwords before they log in again. Alternatively, you can ask users to change their passwords. Vertica hashes all new passwords using the system-level security algorithm.

At the user level, use `ALTER USER` to set the Security_Algorithm user parameter. Changing this parameter at the user level overrides the system-level value:

```sql
=> ALTER USER username SECURITY_ALGORITHM 'MD5' IDENTIFIED BY 'newpassword';
=> ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
```

4. Associate the `v_hash` authentication method with the desired users or user roles, using a `GRANT` statement:

```sql
=> GRANT AUTHENTICATION v_hash to user1, user2, ...;
```

For more information about how these parameters work, see [Hash Authentication Parameters](#).

### Upgrade Considerations for Hash Authentication

For Vertica releases before 7.1, MD5 is the only algorithm used for hashing passwords. In Vertica 7.1, you can use either the MDS algorithm or the more secure SHA-512 algorithm. Before you upgrade, you must consider the following behaviors to avoid problems.

### Upgrade the Client and Server

To implement the more secure SHA-512 algorithm for hashing passwords, you *must* upgrade BOTH the client and the server to Vertica 7.1 or higher. If you upgrade the server but not the client and specify that one or more users store their passwords using SHA-512, the client does not understand hashing with SHA-512. When it sends a message to the server, the server returns an error.
Change Existing Users to SHA-512 Hash Algorithm

When you upgrade from a pre-7.1 database, the user-level parameter Security_Algorithm, is set to 'NONE'. This allows all existing users to continue connecting to the Vertica server and their passwords are hashed using MD5.

If you want one or more users to use the SHA-512 algorithm, set the system-level parameter Security Algorithm to 'SHA512' and change the user passwords.

Use one of three methods to change the user password:

- Manually set the user's user-level security algorithm to 'SHA512'. Then, change the user’s password, as in the following statement:

  ```sql
  => ALTER USER username SECURITY_ALGORITHM 'SHA512' IDENTIFIED BY 'newpassword';
  ```

- Set the user's password to expire immediately as in the following statement. After the password expires, the user responds by changing it.

  ```sql
  => ALTER USER username PASSWORD EXPIRE;
  ```

- Ask the user to change the password.

All new passwords inherit the system-level security algorithm, which is SHA-512.

Passwords

Assign a password to a user to allow that user to connect to the database using password authentication. When the user supplies the correct password a connection to the database occurs.

Vertica stores passwords in an encrypted format to prevent potential theft. However, the transmission of the password to Vertica is in plain text. Thus, it is possible for a "man-in-the-middle" attack to intercept the password.

Implementing Hash Authentication ensures secure login using passwords.

About Password Creation and Modification

You must be a superuser to create passwords for user accounts using the CREATE USER statement. A superuser can set any user account's password.
To add a password, use the ALTER USER statement.

To change a password, use ALTER USER or the vsql \password command.

Users can also change their own passwords.

To make password authentication more effective, Vertica recommends that you enforce password policies that control how often users are forced to change passwords and the required content of a password. You set these policies using Profiles.

Default Password Authentication

When you have not specified any authentication methods, Vertica defaults to using password authentication for user accounts that have passwords.

If you create authentication methods, even for remote hosts, password authentication is disabled. In such cases, you must explicitly enable password authentication. The following commands create the local_pwd authentication method and make it the default for all users.

When you create an authentication method, Vertica enables it automatically:

```
=> CREATE AUTHENTICATION local_pwd METHOD hash' LOCAL;
=> GRANT AUTHENTICATION local_pwd To Public;
```

Profiles

You can set password policies for users by assigning them profiles. You can create multiple profiles to manage the password policies for several categories of users. For example, you might create one profile for interactive users, requiring them to frequently change their passwords. You might create another profile for user accounts that are not required to change passwords.

Defining Assigning Profiles

You create profiles with CREATE PROFILE statement, and change existing profiles with ALTER PROFILE. Both statements let you set one or more profile parameters.

Each profile can specify one or more of the following policies:

- How often users must change their passwords.
- How many times users must change their passwords before they can reuse an old password.
How many times a user can fail to log in before the account is locked.

The required length and content of the password:

- Maximum and minimum number of characters
- Minimum number of capital letters, lowercase letters, digits, and symbols required in a password.

Assigning Profiles

After you define a profile, you can assign it to new and existing users with `CREATE USER` and `ALTER USER`, respectively.

Changes to profile policies for password content—for example, `PASSWORD_MAX_LENGTH` and `PASSWORD_MIN_SYMBOLS`—affect users only when they change their passwords. Vertica does not test existing passwords to verify that they comply with new password requirements. To enforce immediate compliance with new profile requirements, use `ALTER USER...PASSWORD EXPIRE` to force immediate expiration of the user's current password. On the user’s next login, Vertica prompts this user to supply a new password, which must comply with the new policy.

Default Profile

Each database contains a profile named `DEFAULT`. Vertica assigns the default profile to users who are not explicitly assigned a profile. The default profile also sets parameters of non-default profiles in two cases:

- Profile parameters that are not explicitly set by `CREATE PROFILE`
- Parameters that `ALTER PROFILE` sets to `DEFAULT`

All parameters in the default profile are initially set to `unlimited`. You can use `ALTER PROFILE` to change these settings. For example, the following statement modifies the default profile parameter `PASSWORD_MIN_SYMBOLS`. The change requires passwords to contain at least one symbol, such as $, #, @. This change affects all profiles where `PASSWORD_MIN_SYMBOLS` is set to `default`:

```
ALTER PROFILE DEFAULT LIMIT PASSWORD_MIN_SYMBOLS 1;
```

Profile Settings and Client Authentication

The following profile settings affect client authentication methods, such as LDAP or GSS:
- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME

All other profile settings are used only by Vertica to manage its passwords.

See Also

- PROFILES
- Creating a Database Name and Password

Password Guidelines

For passwords to be effective, they must be hard to guess. You need to protect passwords from:

- Dictionary-style, brute-force attacks
- Users who have knowledge of the password holder (family names, birth dates, etc.)

Use Profiles to enforce good password practices (password length and required content). Make sure database users know the password guidelines, and encourage them not to use personal information in their passwords.

For guidelines on creating strong passwords go to Microsoft Tips for Creating a Strong Password.

See Also

- Creating a Database Name and Password

Password Expiration

User profiles control how often users must change their passwords. Initially, the DEFAULT profile is set so that passwords never expire.

Important: Password expiration has no effect on any of the user's current sessions.
Set Password Expiration and Grace Period

You can change the default value to set a password expiration. Alternatively, you can create additional profiles that set time limits for passwords and assign users to them.

When a password expires, the user must change the password on the next login. However, you can set a PASSWORD_GRACE_TIME in any individual user's profile, allowing that user to log in after the expiration. After the password expires, Vertica issues a warning about the password expiration but continues to recognize the password.

After the grace period ends, users must change their passwords to log in, unless they have changed them already in response to the warning.

Expire a Password

You can expire a user's password immediately using the ALTER USER statement's PASSWORD EXPIRE parameter. By expiring a password, you can:

- Force users to comply with a change to password policy.
- Set a new password when a user forgets the old password.

Account Locking

In a profile, you can set a password policy for how many consecutive failed login attempts a user account is allowed before locking. This locking mechanism helps prevent dictionary-style brute-force attempts to guess users' passwords.

Set Account Locking

Set this value using the FAILED_LOGIN_ATTEMPTS parameter using the CREATE PROFILE or ALTER PROFILE statement.

Vertica locks any user account that has more consecutive failed login attempts than the value to which you set FAILED_LOGIN_ATTEMPTS. The user cannot log in to a locked account, even by supplying the correct password.

Unlock a Locked Account

You can unlock accounts in one of two ways, depending on your privileges.
• Manually: If you are a superuser, you can manually unlock the account using the ALTER USER command.

Note: A superuser account cannot be locked, because it is the only user that can unlock accounts. For this reason, choose a very secure password for a superuser account. See Password Guidelines for suggestions.

• Password Lock Time Setting: Specify the number of days until an account unlocks in the PASSWORD_LOCK_TIME parameter of the user's profile. Vertica automatically unlocks the account after the specified number of days has passed. If you set this parameter to UNLIMITED, the user's account is never automatically unlocked, and a superuser must manually unlock it.

Ident Authentication

The Ident protocol, defined in RFC 1413, authenticates a database user with a system user name. To see if that system user can log in without specifying a password, you configure Vertica client authentication to query an Ident server. With this feature, the DBADMIN user can run automated scripts to execute tasks on the Vertica server.

Caution: Ident responses can be easily spoofed by untrusted servers. Use Ident authentication only on local connections, where the Ident server is installed on the same computer as the Vertica database server.

Following the instructions in these topics to install, set up, and configure Ident authentication for your database:

• Installing and Setting Up an Ident Server
• Configuring Ident Authentication for Database Users

Examples

The following examples show several ways to configure Ident authentication.

Allow system_user1 to connect to the database as Vertica vuser1:

=> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
=> ALTER AUTHENTICATION v_ident SET system_users='system_user1';
=> GRANT AUTHENTICATION v_ident to vuser1;
ALTER AUTHENTICATION v_ident ENABLE;

Allow system_user1, system_user2, and system_user3 to connect to the database as vuser1. Use colons (:) to separate the user names:

```sql
-> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
-> ALTER AUTHENTICATION v_ident SET system_users='system_user1:system_user2:system_user3';
-> GRANT AUTHENTICATION v_ident TO vuser1;
-> ALTER AUTHENTICATION v_ident ENABLE;
```

Associate the authentication with Public using a GRANT AUTHENTICATION statement. The users, system_user1, system_user2, and system_user3 can now connect to the database as any database user:

```sql
-> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
-> ALTER AUTHENTICATION v_ident SET system_users='system_user1:system_user2:system_user3';
-> GRANT AUTHENTICATION v_ident to Public;
-> ALTER AUTHENTICATION v_ident ENABLE;
```

Set the system_users parameter to * to allow any system user to connect to the database as vuser1:

```sql
-> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
-> ALTER AUTHENTICATION v_ident SET system_users='*';
-> GRANT AUTHENTICATION v_ident TO vuser1;
-> ALTER AUTHENTICATION v_ident ENABLE;
```

Using a GRANT statement, associate the v_ident authentication with Public to allow system_user1 to log into the database as any database user:

```sql
-> CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
-> ALTER AUTHENTICATION v_ident SET system_users='system_user1';
-> GRANT AUTHENTICATION v_ident to Public;
-> ALTER AUTHENTICATION v_ident ENABLE;
```

### Installing and Setting Up an Ident Server

To use Ident authentication, you must install one or more packages, depending on your operating system, and enable the Ident server on your Vertica server. `oidentd` is an Ident daemon that is compatible with Vertica and compliant with RFC 1413.

**Note:** You can find the source code and installation instructions for `oidentd` at the [oidentd website](#).

To install and configure Ident authentication for use with your Vertica database, follow the appropriate steps for your operating system:
Red Hat 6.x/CentOS 6.x

Install oidentd on Red Hat 6.x or CentOS 6.x by running this command:

$ yum install oidentd

Depending on your configuration, you might receive the following error message:

No package oidentd available.

In this case, you must install the Red Hat/CentOS Extras Repository. Download and install the Extras Repository from the following location: https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm

Red Hat 7.x/CentOS 7.x

Install an Ident server on Red Hat 7.x or CentOS 7.x by installing the authd and xinetd packages:

$ yum install authd
$ yum install xinetd

Ubuntu/Debian

Install oidentd on Ubuntu or Debian by running this command:

$ sudo apt-get install oidentd

SUSE Linux Enterprise Server

Install the pidentd and xinetd RPMs from the following locations:

Post-Installation Steps for Red Hat 6.x/CentOS 6.x and Ubuntu/Debian

After you install `oidentd` on your Red Hat 6.x/CentOS 6.x or Ubuntu/Debian system, continue with the following steps:

1. Verify that the Ident server accepts IPv6 connections to prevent authentication failure. To do so, you must enable this capability. In the script `/etc/init.d/oidentd`, change the line from:

   ```
   exec="/usr/sbin/oidentd"
   ```

   to

   ```
   exec="/usr/sbin/oidentd -a ::"
   ```

   Then, at the Linux prompt, start `oidentd` with `-a ::`.

2. Restart the server with the following command:

   ```
   $ /etc/init.d/oidentd restart
   ```

Post-Installation Steps for Red Hat 7.x/CentOS 7.x and SUSE Linux Enterprise Server

After you install the required packages on your Red Hat 7.x/CentOS 7.x or SUSE Linux Enterprise Server system, continue with the following steps:

1. Enable the auth service in the configuration file located at the following location: `/etc/xinet.d/auth`

   Enter no for the disable option, as this sample configuration file shows.

   ```
   service auth
   {
       disable = no
       socket_type = stream
       wait = no
       user = ident
       cps = 4096 10
   }
   ```
instances = UNLIMITED
server = /usr/sbin/in.authd
server_args = -t60 --xerror --os
}

2. Restart the xinetd service with the following command:

$ service xinetd restart

### Configuring Ident Authentication for Database Users

To configure Ident authentication, take the following steps:

1. Create an authentication method that uses Ident.

   The Ident server must be installed on the same computer as your database, so specify the keyword LOCAL. Vertica requires that the Ident server and database always be on the same computer as the database.

   ```sql
   => CREATE AUTHENTICATION v_ident METHOD 'ident' LOCAL;
   ```

2. Set the Ident authentication parameters, specifying the system users who should be allowed to connect to your database.

   ```sql
   => ALTER AUTHENTICATION v_ident SET system_users='user1:user2:user3';
   ```

3. Associate the authentication method with the Vertica user. Use a GRANT statement that allows the system user `user1` to log in using Ident authentication:

   ```sql
   => GRANT AUTHENTICATION v_ident TO user1;
   ```

### Kerberos Authentication

Kerberos authentication uses the following components to perform user authentication.
Client Package

The Kerberos 5 client package communicates with the KDC server. This package is not included as part of the Vertica Analytics Platform installation. Kerberos software is built into Microsoft Windows. If you are using another operating system, you must obtain and install the client package.

If you do not already have the Kerberos 5 client package on your system, download it from the MIT Kerberos Distribution page. Install the package on each Vertica server and client used in Kerberos authentication, except the KDC itself.

Refer to the Kerberos documentation for installation instructions.

Service Principals

A service principal consists of a host name, a service name, and a realm to which a set of credentials gets assigned (service/hostname@REALM). These credentials connect to the service, which is a host that you connect to over your network and authenticate using the KDC.

See Specify KDC Information and Configure Realms to create the realm name. The host name must match the value supplied by the operating system. Typically this is the fully qualified host name. If the host name part of your principal does not match the value supplied by the operating system, Kerberos authentication fails.

Some systems use a hosts file (/etc/hosts or /etc/hostname) to define host names. A hosts file can define more than one name for a host. The operating system supplies the first entry, so use that in your principal. For example, if your hosts file contains:

```
192.168.1.101  v_vmart_node0001.example.com  v_vmart_node0001
```

then use v_vmart_node0001.example.com as the hostname value.

Note: Depending on your configuration it may be safer to use the fully qualified domain name rather than the hostname.

Configure the following as Kerberos principals:

- Each client (users or applications that connects to Vertica)
- The Vertica server

See the following topics for more information:
Configure Vertica for Kerberos Authentication

Configure Clients for Kerberos Authentication

### Keytab Files

Principals are stored in encrypted keytab files. The keytab file contains the credentials for the Vertica principal. The keytab allows the Vertica server to authenticate itself to the KDC. You need the keytab so that Vertica Analytic Database does not have to prompt for a password.

Create one service principal for each node in your cluster. You can then either create individual keytab files (one for each node containing only that node's principal) or create one keytab file containing all the principals.

- **Create one keytab file with all principals to simplify setup:** all nodes have the same file, making initial setup easier. If you add nodes later you either update (and redistribute) the global keytab file or make separate keytabs for the new nodes. If a principal is compromised it is compromised on all nodes where it is present in a keytab file.

- **Create separate keytab files on each node to simplify maintenance:** Initial setup is more involved as you must create a different file on each node, but no principals are shared across nodes. If you add nodes later you create keytabs on the new nodes. Each node's keytab contains only one principal, the one to use for that node.

### Ticket-Granting Ticket

The Ticket-Granting Ticket (TGT) retrieves service tickets that authenticates users to servers in the domain. Future login requests use the cached HTTP Service Ticket for authentication, unless it has expired as set in the ticket_lifetime parameter in krb5.conf.

### Multi-realm Support

Vertica provides multi-realm support for Kerberos authentication using the SET param=value parameter in `ALTER AUTHENTICATION` with REALM as the parameter:

```
=> ALTER AUTHENTICATION krb_auth_users set REALM='USERS.COM';
=> ALTER AUTHENTICATION krb_auth_realmad set REALM='REALM_AD.COM';
```

This allows you to assign a different realm so that users from another realm can authenticate to Vertica.
Multi-realm support applies to GSS authentication types only. You can have one realm per authentication method. If you have multiple authentication methods, each can have its own realm:

```sql
=> SELECT * FROM client_auth;
auth_oid | auth_name | is_auth_enabled | auth_host_type | auth_host_address | auth_method | auth_parameters | auth_priority
----------|-----------|----------------|----------------|------------------|-------------|----------------|----------------
45035996  | krb001    | True           | HOST           | 0.0.0.0/0        | GSS         |                 | 0              
realm=USERS.COM   | 0          |                |                |                  |             |                 |                
45035997  | user_auth | True           | HOST           | 0.0.0.0/0        | TRUST       |                 | 1000           
realm=REALM_AD.COM | 1          |                |                |                  |             |                 |                
```

Configure Vertica for Kerberos Authentication

Kerberos provides a strong cryptographic authentication against the devices which lets the client & servers to communicate in a more secured manner. It addresses network security problems.

Your system must have one or more Kerberos Key Distribution Centers (KDC) installed and configured. The KDCs must be accessible from every node in your Vertica Analytic Database cluster.

The KDC must support Kerberos 5 using GSS-API. For details, see the MIT Kerberos Distribution Page.

In This Section

Create the Vertica Principals and Keytabs on Linux KDC

Vertica uses service principals for system-level operations. These principals identify the Vertica service and are used as follows:
• Kerberized Vertica clients request access to this service when they authenticate to the database.

• System processes like the Tuple Mover use this identity when they authenticate to external services such as Hadoop.

Create principals and keys as follows:

1. Start the Kerberos 5 database administration utility (kadmin or kadmin.local) to create Vertica principals on a Linux KDC.

   ▶ Use kadmin if you are accessing the KDC on a remote server. If you have access to the Kerberos administrator password, you can use kadmin on any machine where the Kerberos 5 client package is installed. When you start kadmin, the utility prompts you for the Kerberos administrator's password. You might need root privileges on the client to run kadmin.

   ▶ Use kadmin.local if:
     ○ The KDC is on the machine that you are logging in to.
     ○ You have root privileges on that server.

   kadmin.local does not require the administrators login credentials.

   For more information about the kadmin and kadmin.local commands, see the kadmin documentation.

2. Create one service principal for Vertica on each node. The host name must match the value supplied by the operating system. The following example creates the service principal vertica for the node named v_vmart_node0001.example.com:

   ```bash
   $ sudo /usr/kerberos/sbin/kadmin.local
   kadmin.local add_principal vertica/v_vmart_node0001.example.com
   ```

   Repeat the ktadd command once per principal. You can create separate keytabs for each principal user or add them all to a single keytab file (such as krb5.keytab). If you are using a single file, see the documentation for the -glob option in the MIT Kerberos documentation.

   You must have a user principal for each Vertica Analytic Database user that uses Kerberos Authentication. For example:

   ```bash
   $ sudo /usr/kerberos/sbin/kadmin.local
   kadmin.local add_principal [options] VerticaUser1
   ```
3. Copy each keytab file to the /etc folder on the corresponding cluster node. Use the same path and file name on all nodes.

4. On each node, make the keytab file readable by the file owner who is running the database process (typically, the Linux dbadmin user). For example, you can change ownership of the files to dbadmin as follows:

```
$ sudo chown dbadmin *.keytab
```

**Important:** In a production environment, you must control who can access the keytab file to prevent unauthorized users from delegating your server. For more information about delegation (also known as impersonation), see [Technet.Microsoft.com](https://technet.microsoft.com).

After you create a keytab file, you can use the `klist` command to view keys stored in the file:

```
$ sudo /usr/kerberos/bin/klist -ke -t
Keytab name: FILE:/etc/krb5.keytab
KVNO Timestamp Principal
---------- ----------- --------------------------------------------
 4 08/15/2017 7:35:41 vertica/v_mart_node0001.example.com@EXAMPLE.COM (aes256-cts-hmac-sha1-96)
 4 08/15/2017 7:35:41 vertica/v_mart_node0001.example.com@EXAMPLE.COM (aes128-cts-hmac-sha1-96)
```

5. On Vertica run the following to ensure the Kerberos parameters are set correctly:

```
=> select parameter_name, current_value from configuration_parameters where parameter_name like 'Ker%';
```

```
parameter_name | current_value
--------------- | ----------------
KerberosHostname | glCentos5-04.glvpn.com
KerberosKeytabFile | /scratch_b/ga/krbTest/v_krbtest_node0001_catalog/krb5.keytab
KerberosRealm | GLVPN.COM
KerberosTicketDuration | 0
KerberosServiceName | vertica
(4 rows)
```

6. On Vertica, run `KERBEROS_CONFIG_CHECK` to verify the Kerberos configuration.

`KERBEROS_CONFIG_CHECK` verifies the following:

- The existence of the `kinit` and `kb5.conf` files.
- Whether the keytab file exists and is set.
The Kerberos configuration parameters set in the database:
- KerberosServiceName
- KerberosHostname
- KerberosRealm
- Vertica Principal

That Kerberos can read the Vertica keys
That Kerberos can get the tickets for the Vertica principal
That Vertica can initialize the keys with kinit

Creating the Principals and Keytab on Active Directory

Active Directory stores information about members of the Windows domain, including users and hosts.

Vertica uses the Kerberos protocol to access this information in order to authenticate windows users to the vertica database. The Kerberos protocol uses principals to identify users and keytab files to store their cryptographic information. You need to install the keytab files into Vertica to enable the Vertica database to cryptographically authenticate windows users.

This procedure describes:

- The creation of a Vertica service principal.
- Exporting the keytab files for these principals
- Installing the keytab files in the Vertica database. This allows Vertica to authenticate windows users and grant them access to the Vertica database.

1. Create a Windows account (principal) for the Vertica service and one Vertica host for each node/host in the cluster. This procedure creates windows accounts for host verticanode01 and service vertica running on this node.

When you create these accounts, select the following:

- User Cannot change password
- Password never expires
2. If you are using external tables on HDFS that are secured by kerberos authentication, you MUST enable Delegation. To do so, access the Active Directory Users and Computers dialog, right-click on the Windows account (principal) for the Vertica service, and select Delegation. Trust this user for delegation to any service.

3. Run the following command to create the keytab for the host verticanode01.dc.com node/host:

```bash
$ ktpass -out ./host.verticanode01.dc.com.keytab -princ host/verticanode01.dc.com@DC.COM -mapuser verticanode01 -mapop set -pass secret -ptype KRB5_NT_SRV_HST
```

4. Run the following command to create the keytab for the vertica service:

```bash
$ ktpass -out ./vertica.verticanode01dc.com.keytab -princ vertica/verticanode01.dc.com@DC.COM -mapuser vertica -mapop set -pass secret -ptype KRB5_NT_PRINCIPAL
```

For more information about keytab files, see [Technet.Microsoft.com](https://technet.microsoft.com).

5. Run the following commands to ensure the service principal name is mapped correctly. You must run these commands for each node in your cluster:

```bash
$ setspn -L vertica
  Registered ServicePrincipalNamefor CN=vertica,CN=Users,DC=dc,DC=com
  vertica/verticanode01.dc.com

$ setspn -L verticanode01
  Registered ServicePrincipalNamefor CN=verticanode01,CN=Users,DC=dc,DC=com
  host/verticanode01.dc.com
```


7. Combine the keytab files into a single keytab:

```bash
[release@vertica krbTest]$ /usr/kerberos/sbin/ktutil
ktutil:  rkt host.verticanode01.dc.com.keytab
ktutil:  rkt vertica.verticanode01.dc.com.keytab
ktutil:  list
slot KVNO Principal
```

**Note:** You can deselect Password never expires, but if you change these user passwords you must recreate the keytab files and reinstall them into Vertica. This includes repeating the entire procedure.
This creates a single keytab file that contains the server principal for authentication.

8. Copy the new keytab file to the catalog directory. For example:

```
$ cp verticanode01.dc.com.keytab /home/dbadmin/VMart/v_vmart_nodennnn_catalog
```

9. Test the keytab file’s ability to retrieve a ticket to ensure it works from the Vertica node:

```
$ kinit vertica/verticanode01.dc.com -k -t verticanode01.dc.com.keytab
$ klist
```

```
Ticket cache: KFILE:/tmp/krb_ccache_1003
Default principal: vertica/verticanode01.dc.com@DC.COM

Valid starting Expires Service principal
04/08/2017 13:35:25 04/08/2017 23:35:25 krbtgt/DC.COM@DC.COM
renew until 04/15/2017 14:35:25
```

When the ticket expires or not automatically retrieved you need to manually run the kinit command. See Get the Kerberos Ticket and Authenticate Vertica.

10. Set the right permissions and ownership on the keytab files:

```
$ chmod 600 verticanode01.dc.com.keytab
$ chown dbadmin:verticadba verticanode01.dc.com.keytab
```

11. Set the following Kerberos Configuration Parameters using ALTER DATABASE to inform Vertica about the Kerberos principal:

```
KerberosKeytabFile=<CATALOGDIR>/verticanode01.dc.com.keytab
KerberosRealm=DC.COM
KerberosServiceName=vertica
KerberosTicketDuration = 0
KerberosHostname=verticanode01.dc.com
```

12. Restart the Vertica server.

13. Test your Kerberos setup as follows to ensure that all clients use the gss authentication method.

   From Vertica:
CREATE USER windowsuser1;
CREATE USER v_kerberos method 'gss' host '0.0.0.0/0';
ALTER AUTHENTICATION v_kerberos enable;
GRANT AUTHENTICATION v_kerberos to windowsuser1;

From OS command line:

$ kinit windowsuser1
$ vsql -U windowsuser1 -k vertica -K verticanode01.dc.com -h verticanode01.dc.com -c "select client_authentication_name, authentication_method from sessions;"

14. Run KERBEROS_CONFIG_CHECK to verify the Kerberos configuration. KERBEROS_CONFIG_CHECK verifies the following:

- The existence of the kinit and kb5.conf files.
- Whether the keytab file exists and is set
- The Kerberos configuration parameters set in the database:
  - KerberosServiceName
  - KerberosHostname
  - KerberosRealm
  - Vertica Principal
- That Kerberos can read the Vertica keys
- That Kerberos can get the tickets for the Vertica principal
- That Vertica can initialize the keys with kinit
Get the Kerberos Ticket and Authenticate Vertica

If your organization uses Kerberos as part of the login process, Kerberos tickets are automatically retrieved upon login. Otherwise, you need to run `kinit` to retrieve the Kerberos ticket.

The following example shows how to retrieve the ticket and authenticate Vertica Analytic Database with the KDC using the `kinit` command. EXAMPLE.COM is the realm name. You must use the realm name with your username to retrieve a Kerberos ticket. See Specify KDC Information and Configure Realms.

```
$ kinit
Password for principal_user@EXAMPLE.COM: kpasswd
```

You are prompted for the password of the principal user name created when you created the principals and keytabs (see Create the Vertica Principals and Keytabs on Linux KDC).

The Kerberos ticket gets cached for a pre-determined length of time. See Ticket Management in the Kerberos documentation for more information on setting expiration parameters.

Upon expiration, you need to run the `kinit` command again to retrieve another Kerberos ticket.

Configure Clients for Kerberos Authentication

Each supported platform has a different security framework. Thus, the steps required to configure and authenticate against Kerberos differ among clients.

On the server side, you construct the Vertica Kerberos service name principal using this format:

```
Kerberos_Service_Name/Kerberos_Host_Name@Kerberos_REALM
```

For each client, the GSS libraries require the following format for the Vertica service principal:

```
Kerberos_Service_Name@Kerberos_Host_Name
```

You can omit the realm portion of the principal because GSS libraries use the realm name of the configured default (Kerberos_REALM) realm.

For information about client connection strings, see the following topics in Connecting to Vertica:
In This Section

- Configure ODBC and vsql Clients on Linux, HP-UX, AIX, MAC OSX, and Solaris
- Configure ODBC and vsql Clients on Windows and ADO.NET
- Configure JDBC Clients on all Platforms

Configure ODBC and vsql Clients on Non-Windows Platforms

To configure an ODBC or vsql client on Linux, HP-UX, AIX, MAC OSX, or Solaris, you must first install the Kerberos 5 client package. See Configuring Kerberos Authentication.

After you install the Kerberos 5 client package, you must provide clients with a valid Kerberos configuration file (krb5.conf). To communicate with the KDC, each client participating in Kerberos authentication must have a valid, identically configured krb5.conf file. The default location for the Kerberos configuration file is /etc/krb5.conf.

Tip: To enforce consistency among clients, Vertica Analytic Database, and the KDC, copy the /etc/krb5.conf file from the KDC to the client's/etc directory.

The Kerberos configuration (krb5.conf) file contains Kerberos-specific information, including:

- How to reach the KDC
- Default realm name
- Domain
- Path to log files

Note: A few scenarios exist in which the Vertica server principal name might not match the host name in the connection string. See Troubleshooting Kerberos Authentication for more information.
- DNS lookup
- Encryption types to use
- Ticket lifetime

The default location for the Kerberos configuration file is `/etc/krb5.conf`. When configured properly, the client can authenticate with Kerberos and retrieve a ticket through the `kinit` utility (see Acquire an ODBC Authentication Request and Connection below). Likewise, the server can then use `ktutil` to store its credentials in a keytab file.

Authenticating ODBC and vsql Clients Requests and Connections on Non-Windows Platforms

ODBC and vsql use the client's ticket established by `kinit` to perform Kerberos authentication. These clients rely on the security library's default mechanisms to find the ticket file and the Kerberos configuration file.

To authenticate against Kerberos, call the `kinit` utility to obtain a ticket from the Kerberos KDC server. The following two examples show how to send the ticket request using ODBC and vsql clients.

**Acquire an ODBC Authentication Request and Connection**

1. On an ODBC client, acquire a ticket for the `kuser` user by calling the `kinit` utility.

   ```
   $ kinit kuser@EXAMPLE.COM
   Password for kuser@EXAMPLE.COM:
   ```

2. Connect to Vertica, and provide the principals in the connection string:

   ```
   char outStr[100];
   SQLLEN len;
   SQLDriverConnect(handle, NULL, "Database=VMart;User=kuser;
   Server=myserver.example.com;Port=5433;KerberosHostname=vcluster.example.com",
   SQL_NT5, outStr, &len);
   ```

**Acquire a vsql Authentication Request Connection**

If the `vsql` client is on the same machine you are connecting to, `vsql` connects through a UNIX domain socket. This connection bypasses Kerberos authentication. When you authenticate
with Kerberos, especially if the client authentication method is configured as 'local', you must include the -h hostname option. See Command Line Options in Connecting to Vertica.

1. On the vsql client, call the `kinit` utility:

   ```bash
   $ kinit kuser@EXAMPLE.COM
   Password for kuser@EXAMPLE.COM:
   ```

2. Connect to Vertica, and provide the host and user principals in the connection string:

   ```bash
   $ ./vsql -K vcluster.example.com -h myserver.example.com -U kuser
   Welcome to vsql, the Vertica Analytic Database
   interactive terminal.
   Type: \h or \? for help with vsql commands
   \g or terminate with semicolon to execute query
   \q to quit
   ```

   In the future, when you log in to vsql as `kuser`, vsql uses your cached ticket without prompting you for a password.

Verify the Authentication Method

You can verify the authentication method by querying the SESSIONS system table:

```sql
=> SELECT authentication_method FROM sessions;
   authentication_method
   ---------------------
   GSS-Kerberos
   (1 row)
```

See Also

- Data Source Name (DSN) Connection Properties in Connecting to Vertica
- (vsql) Command-Line Options in Connecting to Vertica

Configure ADO.NET, ODBC, and vsql Clients on Windows

The Vertica client drivers support the Windows SSPI library for Kerberos authentication. Windows Kerberos configuration is stored in the registry.

You can choose between two different setup scenarios for Kerberos authentication on ODBC and vsql clients on Windows and ADO.NET:
Windows KDC on Active Directory with Windows Built-in Kerberos Client and Vertica

Linux KDC with Windows Built-in Kerberos Client and Vertica

Windows KDC on Active Directory with Windows Built-in Kerberos Client and Vertica

Kerberos authentication on Windows is commonly used with Active Directory, Microsoft’s enterprise directory service/Kerberos implementation. Typically your organization's network or IT administrator performs the setup.

Windows clients have Kerberos authentication built into the authentication process. You do not need any additional software.

Your login credentials authenticate you to the Kerberos server (KDC) when you:

- Log in to Windows from a client machine
- Use a Windows instance that has been configured to use Kerberos through Active Directory

To use Kerberos authentication on Windows clients, log in as REALM\user.

Important: When you use the ADO.NET driver to connect to Vertica, you can optionally specify IntegratedSecurity=true in the connection string. This informs the driver to authenticate the calling user against the user's Windows credentials. As a result, you do not need to include a user name or password in the connection string. Any user=<username> entry to the connection string is ignored.

Linux KDC with Windows Built-in Kerberos Client and Vertica

A simple, but less common scenario is to configure Windows to authenticate against a non-Windows KDC. In this implementation, you use the ksetup utility to point the Windows operating system native Kerberos capabilities at a non-Active Directory KDC. By logging in to Windows, you obtain a ticket-granting ticket, similar to the Active Directory implementation. However, in this case, Windows is internally communicating with a Linux KDC. See the Microsoft Windows Server Ksetup page for more information.

When a database/windows user logs into their Windows machine (or after performing a kinit on Windows) the Kerberos ticket MUST have ok_as_delegate and forwardable flag set to be able to access webhdfs based external tables as follows:

```bash
$ CMD \> klist
#2> Client: release @ VERTQA.LOCAL
Server: vertica/vqatest108.verticacorp.com @ VERTQA.LOCAL
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
```
Note: The Ticket Flags setting above must contain ok_as_delegate and forwardable entries. For information on these parameters see Kerberos documentation.

Configure Windows Clients for Kerberos Authentication

Depending on which implementation you want to configure, refer to one of the following pages on the Microsoft Server website:

- To set up Windows clients with Active Directory, refer to Step-by-Step Guide to Kerberos 5 (krb5 1.0) Interoperability.

- To set up Windows clients with the ksetup utility, refer to the Ksetup page.

Authenticate and Connect Clients

The KDC can authenticate both an ADO.NET and a vsql client.

Note: Use the fully-qualified domain name as the server in your connection string; for example, use host.example.com instead of just host. That way, if the server moves location, you do not have to change your connection string.

Verify an ADO.NET Authentication Request and Connection

This example shows how to use the IntegratedSecurity=true, setting to specify that the ADO.NET driver authenticate the calling user's Windows credentials:

```java
VerticaConnection conn = new VerticaConnection("Database=VMart;Server=host.example.com;Port=5433;IntegratedSecurity=true;KerberosServiceName=vertica;KerberosHostname=vcluster.example.com");
conn.open();
```
Verify a vsq1 Authentication Request and Connection

1. Log in to your Windows client, for example, as EXAMPLE\kuser.

2. Run the vsq1 client and supply the connection string to Vertica:

```bash
C:\Users\kuser\Desktop>vsql.exe -h host.example.com -K vcluster -U kuser
```

Welcome to vsq1, the Vertica Analytic Database interactive terminal.
Type: \h or \? for help with vsq1 commands
\g or terminate with semicolon to execute query
\q to quit

See Also

- Kerberos Client/Server Requirements
- vsq1 Command Line Options in Connecting to Vertica
- ADO.NET Connection Properties in Connecting to Vertica

Configure JDBC Clients on All Platforms

Kerberos authentication on JDBC clients uses Java Authentication and Authorization Service (JAAS) to acquire the initial Kerberos credentials. JAAS is an API framework that hides platform-specific authentication details and provides a consistent interface for other applications.

You specify the client login process through the JAAS Login Configuration File. This file contains options that specify the authentication method and other settings to use for Kerberos. A class called the LoginModule defines valid options in the configuration file.

The JDBC client principal is crafted as jdbc-username@server-from-connection-string.

Implement the LoginModule

Vertica recommends that you use the JAAS public class com.sun.security.auth.module.Krb5LoginModul provided in the Java Runtime Environment (JRE).

The Krb5LoginModule authenticates users using Kerberos protocols and is implemented differently on non-Windows and Windows platforms:
On non-Windows platforms: The Krb5LoginModule defers to a native Kerberos client implementation. Thus, you can use the same /etc/krb5.conf setup as you use to configure ODBC and vsql clients on Linux, HP-UX, AIX, MAC OSX, and Solaris platforms.

On Windows platforms: The Krb5LoginModule uses a custom Kerberos client implementation bundled with the Java Runtime Environment (JRE). Windows settings are stored in a %WINDIR%\krb5.ini file, which has similar syntax and conventions to the non-Windows krb5.conf file. You can copy a krb5.conf from a non-Windows client to %WINDIR%\krb5.ini.

You can find documentation for the LoginModules in the com.sun.security.auth package, and on the Krb5LoginModule web page.

Create the JAAS Login Configuration

The JAASConfigName connection property identifies a specific configuration within a JAAS configuration that contains the Krb5LoginModule and its settings. The JAASConfigName setting lets multiple JDBC applications with different Kerberos settings coexist on a single host. The default configuration name is verticajdbc.

Important: Carefully construct the JAAS login configuration file. If syntax is incorrect, authentication fails.

You can configure JAAS-related settings in the java.security master security properties file. This file resides in the lib/security directory of the JRE. For more information, see Appendix A in the Java™ Authentication and Authorization Service (JAAS) Reference Guide.

Create a JDBC Login Context

The following example shows how to create a login context for Kerberos authentication on a JDBC client. The client uses the default JAASConfigName of verticajdbc and specifies that:

- The ticket-granting ticket will be obtained from the ticket cache
- The user will not be prompted for a password if credentials cannot be obtained from the cache, keytab file, or through a shared state.

```java
verticajdbc {
    com.sun.security.auth.module.Krb5LoginModule
        required
        useTicketCache=true
        doNotPrompt=true;
};
```
JDBC Authentication Request and Connection

You can configure the Krb5LoginModule to use a cached ticket or keytab. The driver can also acquire a ticket or keytab automatically if the calling user provides a password.

In the preceding example, the login process uses a cached ticket and does not prompt for a password because both useTicketCache and doNotPrompt are set to true. If doNotPrompt=false and you provide a user name and password during the login process, the driver provides that information to the LoginModule. The driver then calls the kinit utility on your behalf.

1. On a JDBC client, call the kinit utility to acquire a ticket:

   ```bash
   $ kinit kuser@EXAMPLE.COM
   ```

   If you prefer to use a password instead of calling the kinit utility, see the next section.

2. Connect to Vertica:

   ```java
   Properties props = new Properties();
   props.setProperty("user", "kuser");
   props.setProperty("KerberosServiceName", "vertica");
   props.setProperty("KerberosHostName", "vcluster.example.com");
   props.setProperty("JAASConfigName", "verticajdbc");
   Connection conn = DriverManager.getConnection("jdbc:vertica://myserver.example.com:5433/VMart", props);
   ```

Have the Driver Acquire a Ticket

Sometimes, you may want to bypass calling the kinit utility yourself but still use encrypted, mutual authentication. In such cases, you can optionally pass the driver a clear text password to acquire the ticket from the KDC. The password is encrypted when sent across the network. For example, useTicketCache and doNotPrompt are both false in the following example. Thus, the calling user's credentials are not obtained through the ticket cache or keytab.

```bash
$ verticajdbc {
   com.sun.security.auth.module.Krb5LoginModule
   required
   useTicketCache=false
   doNotPrompt=false;
};
```

The preceding example demonstrates the flexibility of JAAS. The driver no longer looks for a cached ticket, and you do not have to call kinit. Instead, the driver takes the password and user name and calls kinit on your behalf.
See Also

- Kerberos Client/Server Requirements
- JDBC Connection Properties in Connecting to Vertica
- Java™ Authentication and Authorization Service (JAAS) Reference Guide (external website)

Troubleshooting Kerberos Authentication

These tips can help you avoid issues related to Kerberos authentication with Vertica Analytic Database and to troubleshoot any problems that occur.

JDBC Client Authentication Fails

If Kerberos authentication fails on a JDBC client, check the JAAS login configuration file for syntax issues. If syntax is incorrect, authentication fails.

Working Domain Name Service (DNS) Not Configured

Verify that the DNS entries and the system host file (/etc/hosts or /etc/hostname) on the network are all properly configured for your environment. If you are using a fully qualified domain name, ensure that is properly configured as well. Refer to the Kerberos documentation for your platform for details.

System Clocks Out of Sync

All Systems Except Red Hat 7/CentOS 7

System clocks in your network must remain in sync for Kerberos authentication to work properly. To do so:

1. Install NTP on the Kerberos server (KDC).
2. Install NTP on each server in your network.
3. Synchronize system clocks on all machines that participate in the Kerberos realm within a few minutes of the KDC and each other
Clock skew can be a problem on Linux virtual machines that need to sync with the Windows Time Service. Use the following steps to keep time in sync:

1. Using any text editor, open /etc/ntp.conf.

2. Under the Undisciplined Local Clock section, add the IP address for the Vertica Analytic Database server. Then, remove existing server entries.

3. Log in to the server as root, and set up a cron job to sync time with the added IP address every half hour, or as often as needed. For example:

   ```
   # 0 */2 * * * /etc/init.d/ntpd restart
   ```

4. Alternatively, run the following command to force clock sync immediately:

   ```
   $ sudo /etc/init.d/ntpd restart
   ```

For more information, see Enabling Network Time Protocol (NTP) in Installing Vertica and the Network Time Protocol website.

**Red Hat 7/CentOS 7 Systems**

In Red Hat 7/CentOS 7, ntpd is deprecated in favor of chrony. To keep system clocks in your network in sync for Kerberos authentication to work properly, do the following:

1. Install chrony on the Kerberos server (KDC).

2. Install chrony on each server in your network.

3. Synchronize system clocks on all machines that participate in the Kerberos realm within a few minutes of the KDC and each other.

**Clock Skew on Linux Virtual Machines**

Clock skew can be problematic on Linux virtual machines that need to sync with the Windows Time Service. Try the following to keep time in sync:

1. Using any text editor, open /etc/chrony.conf.

2. Under the Undisciplined Local Clock section, add the IP address for the Vertica Analytic Database server. Then, remove existing server entries.
3. Log in to the server as root, and set up a cron job to sync time with the added IP address every half hour, or as often as needed. For example:

```
# 0 */2 * * * systemctl start chronyd
```

4. Alternatively, run the following command to force clock sync immediately:

```
$ sudo systemctl start chronyd
```

For more information, see Set Up Time Synchronization in Installing Vertica and the Red Hat chrony guide.

**Kerberos Ticket Is Valid, But Hadoop Access Fails**

Vertica uses Kerberos tickets to obtain Hadoop tokens. It then uses the Hadoop tokens to access the Hadoop data. Hadoop tokens expire after a period of time, so Vertica periodically refreshes them. However, if your Hadoop cluster is set to expire tokens frequently, it is possible that tokens might not be refreshed in time. If the token expires, you cannot access data.

Setting the HadoopFSTokenRefreshFrequency configuration parameter allows you to specify how often Vertica should refresh the token. Specify this value, in seconds, to be smaller than the expiration period set for Hadoop. For example:

```
=> ALTER DATABASE exampledb SET HadoopFSTokenRefreshFrequency = '86400';
```

**Encryption Algorithm Choices**

Kerberos is based on symmetric encryption. Be sure that all Kerberos parties used in the Kerberos realm agree on the encryption algorithm to use. If they do not agree, authentication fails. You can review the exceptions in the vertica.log.

On a Windows client, be sure the encryption types match the types set on Active Directory. See Configure Vertica for Kerberos Authentication.

Be aware that Kerberos is used only for securing the login process. After the login process completes, by default, information travels between client and server without encryption. If you want to encrypt traffic, use SSL. For details, see Implementing SSL.
Kerberos Passwords Not Recognized

If you change your Kerberos password, you must re-create all of your keytab files.

Using the ODBC Data Source Configuration Utility

On Windows vsql clients, you may choose to use the ODBC Data Source Configuration utility and supply a client Data Source. If so, be sure you enter a Kerberos host name in the Client Settings tab to avoid client connection failures with the Vertica Analytic Database server.

Authentication Failure in Backup, Restore, or Admin Tools

This problem can arise in configurations where each Vertica node uses its own Kerberos principal. (This configuration is recommended.) When using vbr or admintools you might see an error such as the following:

```
$ vsql: GSSAPI continuation error: Miscellaneous failure
GSSAPI continuation error: Server not found in Kerberos database
```

Backup/restore and the admin tools use the value of KerberosHostname, if it is set, in the Kerberos principal used to authenticate. The same value is used on all nodes. If you have defined one Kerberos principal per node, as recommended, this value does not match. To correct this, unset the KerberosHostname parameter:

```
=> ALTER DATABASE mydb CLEAR KerberosHostname;
```

Server's Principal Name Does Not Match Host Name

This problem can arise in configurations where a single Kerberos principal is used for all nodes. OpenText recommends against using a single Kerberos principal for all nodes. Instead, use one principal per node and do not set the KerberosHostname parameter.

In some cases during client connection, the Vertica server's principal name might not match the host name in the connection string. (See also Using the ODBC Data Source Configuration Utility in this topic.)

On Windows vsql clients, you may choose to use the ODBC Data Source Configuration utility and supply a client Data Source. If so, be sure you enter a Kerberos host name in the Client Settings tab to avoid client connection failures with the Vertica Analytic Database server.
On ODBC, JDBC, and ADO.NET clients, set the host name portion of the server's principal using the KerberosHostName connection string.

Tip: On vsq1 clients, you set the host name portion of the server's principal name using the -K KRB HOST command-line option. The default value is specified by the -h switch, which is the host name of the machine on which the Vertica server is running. -K is equivalent to the drivers' KerberosHostName connection string value.

For details, see Command Line Options in Connecting to Vertica.

Principal/Host Mismatch Issues and Resolutions

This section lists potential issues that may occur if the principal and host are mismatched.

- The KerberosHostName configuration parameter has been overridden.

  For example, consider the following connection string:

  ```
  jdbc:vertica://v_vmart_node0001.example.com/vmart?user=kuser
  ```

  Because the this connection string includes no explicit KerberosHostName parameter, the driver defaults to the host in the URL (v_vmart_node0001.example.com). If you overwrite the server-side KerberosHostName parameter as “abc”, the client generates an incorrect principal.

  To resolve this issue, explicitly set the client’s KerberosHostName to the connection string, as in this example:

  ```
  jdbc:vertica://v_vmart_node0001.example.com/vmart?user=kuser&kerberoshostname=abc
  ```

- Connection load balancing is enabled, but the node against which the client authenticates might not be the node in the connection string.

  In this situation, consider changing all nodes to use the same KerberosHostName setting. When you use the default to the host that was originally specified in the connection string, load balancing cannot interfere with Kerberos authentication.

- You have a DNS name that does not match the Kerberos host name.

  For example, imagine a cluster of six servers, where you want hr-servers and finance-servers to connect to different nodes on the Vertica Analytic Database cluster. Kerberos
authentication, however, occurs on a single (the same) KDC. In the following example, the Kerberos service host name of the servers is server.example.com.

Suppose you have the following list of example servers:

<table>
<thead>
<tr>
<th>Server Name</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>server1.example.com</td>
<td>192.16.10.11</td>
</tr>
<tr>
<td>server2.example.com</td>
<td>192.16.10.12</td>
</tr>
<tr>
<td>server3.example.com</td>
<td>192.16.10.13</td>
</tr>
<tr>
<td>server4.example.com</td>
<td>192.16.10.14</td>
</tr>
<tr>
<td>server5.example.com</td>
<td>192.16.10.15</td>
</tr>
<tr>
<td>server6.example.com</td>
<td>192.16.10.16</td>
</tr>
</tbody>
</table>

Now, assume you have the following DNS entries:

<table>
<thead>
<tr>
<th>Name</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>finance-servers.example.com</td>
<td>192.168.10.11, 192.168.10.12, 192.168.10.13</td>
</tr>
<tr>
<td>hr-servers.example.com</td>
<td>192.168.10.14, 192.168.10.15, 192.168.10.16</td>
</tr>
</tbody>
</table>

When you connect to finance-servers.example.com, specify:

- Kerberos -h host name option as server.example.com
- -K host option for hr-servers.example.com

For example:

```
$ vsql -h finance-servers.example.com -K server.example.com
```

- You do not have DNS set up on the client machine, so you must connect by IP only.

To resolve this issue, specify:

- Kerberos -h host name option for the IP address
- -K host option for server.example.com

For example:

```
$ vsql -h 192.168.1.12 -K server.example.com
```

- There is a load balancer involved (Virtual IP), but there is no DNS name for the VIP.

Specify:
**Kerberos**

- **-h** host name option for the Virtual IP address
- **-K** host option for server.example.com

For example:

```
$ vsq1 -h <virtual IP> -K server.example.com
```

- You connect to Vertica using an IP address, but there is no host name to construct the Kerberos principal name.

  Provide the instance or host name for the Vertica as described in [Inform About the Kerberos Principal](#).

- You set the server-side **KerberosHostName** configuration parameter to a name other than the Vertica node's host name, but the client cannot determine the host name based on the host name in the connection string alone.

  **Rename the KerberosHostName**

  Rename the **KerberosHostName** to match the name of the Vertica node's host name. For more information, see the following topics in Connecting to Vertica:

  - **ODBC DSN Parameters**
  - **JDBC Connection Properties**
  - **ADO.NET Connection Properties**

**LDAP Authentication**

Lightweight Directory Access Protocol (LDAP) is an authentication method that works like password authentication. The main difference is that the LDAP method authenticates clients trying to access your Vertica database against an LDAP or Active Directory server. Use LDAP authentication when your database needs to authenticate a user with an LDAP or Active Directory server.

Before you configure LDAP authentication for your Vertica database, review the following information:
LDAP Prerequisites and Definitions

Prerequisites

Before you configure LDAP authentication for your Vertica database you must have:

- IP address and host name for the LDAP server. Vertica supports IPv4 and IPv6 addresses.
- Your organization's Active Directory information.
- A service account for search and bind.
- Administrative access to your Vertica database.
- open-ldap-tools package installed on at least one node. This package includes ldapsearch.

Definitions

The following definitions are important to remember for LDAP authentication:

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>IP address or host name of the LDAP server. Vertica supports IPv4 and IPv6 addresses. For more information, see IPv4 and IPv6 for Client Authentication.</td>
</tr>
<tr>
<td>Common name (CN)</td>
<td>Depending on your LDAP environment, this value can be either the username or the first and last name of the user.</td>
</tr>
<tr>
<td>Parameter name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Domain component (DC)</td>
<td>Comma-separated list that contains your organization's domain component broken up into separate values, for example: dc=vertica, dc=com</td>
</tr>
<tr>
<td>Distinguished name (DN)</td>
<td>domain.com. A DN consists of two DC components, as in &quot;DC=example, DC= com&quot;.</td>
</tr>
<tr>
<td>Organizational unit (OU)</td>
<td>Unit in the organization with which the user is associated, for example, Vertica Users.</td>
</tr>
<tr>
<td>sAMAccountName</td>
<td>An Active Directory user account field. This value is usually the attribute to be searched when you use bind and search against the Microsoft Active Directory server.</td>
</tr>
<tr>
<td>UID</td>
<td>A commonly used LDAP account attribute used to store a username.</td>
</tr>
<tr>
<td>Bind</td>
<td>LDAP authentication method that allows basic binding using the DN.</td>
</tr>
<tr>
<td>Search and bind</td>
<td>LDAP authentication method that must log in to the LDAP server to search on the specified attribute.</td>
</tr>
<tr>
<td>Service account</td>
<td>An LDAP user account that can be used to log in to the LDAP server during bind and search. This account's password is usually shared.</td>
</tr>
<tr>
<td>Anonymous binding</td>
<td>Allows a client to connect and search the directory (search and bind) without needing to log in.</td>
</tr>
<tr>
<td>ldapsearch</td>
<td>A command-line utility to search the LDAP directory. It returns information that you use to configure LDAP search and bind.</td>
</tr>
<tr>
<td>basedn</td>
<td>Distinguished name where the directory search should begin.</td>
</tr>
<tr>
<td>binddn</td>
<td>Domain name to find in the directory search.</td>
</tr>
<tr>
<td>search_attribute</td>
<td>Text to search for to locate the user record. The default is UID.</td>
</tr>
</tbody>
</table>

**LDAP Parameters**

There are several parameters that you need to configure for LDAP authentication.
General LDAP Parameters

Use the following parameters to configure for either LDAP bind or LDAP bind and search:

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
</table>
| host           | LDAP server URI in the following format: `schema://host:optional_port`  
`schema` is either `ldap` (for LDAP/Active Directory) or `ldaps` (for secure LDAP/Active Directory). |
| starttls       | Optional parameter that defines StartTLS behavior:  
- `soft`—If the server does not support TLS, continue authenticating the user in plain text. This value is equivalent to the `-Z` option in `ldapsearch`.  
- `hard`—If server does not support TLS, authentication should fail. This value is equivalent to the `-ZZ` in `ldapsearch`.  
  
Using ldaps is equivalent to `starttls='hard'`. However, if you use them together in the same connection string, authentication fails and the following error appears:  

```
FATAL 3846: LDAP authentication failed for user "<user_name>"
```

| 1dap_continue | When set to yes, this parameter allows a connection retry when a user not found error occurs during the previous connection attempt.  
For any other failure error, the system automatically retries the connection. |

LDAP Bind Parameters

Use the following parameters when authenticating with LDAP bind to create the bind name string. For more information see [Workflow for Configuring LDAP Bind](#).
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binddn_prefix</td>
<td>First half of the bind string.</td>
</tr>
</tbody>
</table>
| binddn_suffix  | Second half of bind string.  
You must use the binddn_prefix and binddn_suffix together. 
In the following example, the bind name becomes `cn=<user_login_name>;ou=vertica_users;dc=verticacorp;dc=com`.  
`=> ALTER AUTHENTICATION auth_method_name SET binddn_prefix='cn=',binddn_suffix=''; ou=vertica_users;dc=verticacorp;dc=com';` |
| domain_prefix | The domain where to find the user name. 
In the following example, the bind name is `verticacorp/<user_login_name>`  
`ALTER AUTHENTICATION auth_method_name SET domain_prefix='Example';` |
| email_suffix | The part of an email address that comes after the @ sign.  
In the following example, the bind name becomes `<user_login_name>@example.com`  
`=> ALTER AUTHENTICATION auth_method_name SET email_suffix='Example.com';` |

To create the bind name string, you must provide one of the following:

- Both binddn_prefix and binddn_suffix
- domain_name
- email_suffix

Otherwise, Vertica performs a bind and search operation instead of a bind operation.

### LDAP Search and Bind Parameters

Use the following parameters when authenticating with LDAP search and bind. For more information see [Workflow for Configuring LDAP Search and Bind](#).
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basedn</td>
<td>Base DN for search.</td>
</tr>
<tr>
<td>binddn</td>
<td>Bind DN. Domain name to find in the directory search.</td>
</tr>
<tr>
<td>bind_password</td>
<td>Bind password. Required if you specify a binddn.</td>
</tr>
<tr>
<td>search_attribute</td>
<td>Optional attribute to search for on the LDAP server.</td>
</tr>
</tbody>
</table>

The following example shows how to set these three attributes. In this example, it sets:

- `binddn` to `cn=Manager,dc=example,dc=com`
- `bind_password` to `secret`
- `search_attribute` to `cn`

```
=> ALTER AUTHENTICATION auth_method_name SET host='ldap://example13', basedn='dc=example,dc=com', binddn='cn=Manager,dc=example,dc=com', bind_password='secret', search_attribute='cn';
```

The `binddn` and `bind_password` parameters are optional. If you omit them, Vertica performs an anonymous search.

**Using LDAP Over SSL/TLS**

Vertica supports Transport Layer Security (TLS) for client authentication. TLS uses OpenSSL 0.9.8za and SSL v3/Transport Layer Security (TLS) 1.0 protocol.

The terms SSL and TLS are often used interchangeably. TLS is the successor to SSL and offers greater security. The original SSL standard was renamed TLS at the time it became open source. The introduction of TLS began with version 1, which is essentially equal to SSL 3. You use openssl commands to create certificates and keys and TLS syntax to create an authentication method.

For more information see the [Information Security website](#).

You use `ALTER AUTHENTICATION` to specify LDAP and SSL/TLS parameters. If you specify a host URL that starts with `ldaps`, the Vertica server authenticates using SSL/TLS on the specified port or on the secure LDAPS port (636).
ldaps://abc.dc.com

If the LDAP server does not support SSL on that port, authentication fails.

If you specify a host URL that starts with ldap and set the LDAP starttls parameter, the Vertica server sends a StartTLS request. This request determines if the LDAP server supports TLS on the specified port or on the default LDAP port (389).

```sql
=> ALTER AUTHENTICATION Ldap1 SET host='ldaps://abc.dc.com', binddn_prefix='CN=',
   binddn_suffix=',OU=Unit2,DC=dc,DC=com', basedn='dc=dc,dc=com',
   tls_cacert='~/home/dc.com.ca.cer', tls_reqcert='never';
```

If the LDAP server does not support TLS on that port, the result depends on the value of the starttls parameter:

- **starttls = hard**: The Vertica server terminates the authentication process.
- **starttls = soft**: The Vertica server proceeds with the authentication but does not use TLS.

To configure LDAP over SSL/TLS, use the following configuration parameters:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_REQCERT</td>
<td>hard—If the client does not provide a certificate, or provides an invalid certificate, it cannot connect. This is the default behavior. never—The client does not request or verify a certificate. allow—If the client does not provide a certificate or provides an invalid certificate, it can connect anyway. try—If the client does not provide a certificate, they can connect. If the client provides an invalid certificate, they cannot connect.</td>
</tr>
</tbody>
</table>
| TLS_CADIR      | Path to the folder with the CA certificates. For example: 
ALTER AUTHENTICATION Ldap1 SET TLS_CADIR ='/scratch_b/qa/vertica/QA/VT_Scenario/V_SEC/'; |
| TLS_CACERT     | Path to the CA certificate. For example: 
ALTER AUTHENTICATION Ldap1 SET TLS_CACERT ='/scratch_b/qa/vertica/QA/VT_Scenario/V_SEC/dc.com.ca.cer'; |

If you do not provide one or more of these parameters, the LDAP server checks to see if the LDAPNOINIT environment variable points to the ldap.conf file. If it does, the server uses the
parameters specified in the ldap.conf file. If the LDAP server cannot find the ldap.conf file, authentication fails.

The following example shows how to specify the TLS parameters and the LDAP parameters when configuring LDAP over SSL/TLS:

```sql
=> CREATE AUTHENTICATION LDAP1 METHOD 'ldap' HOST :clientIP = '172.16.65.177';
=> GRANT AUTHENTICATION ldap1 TO user1;
=> ALTER AUTHENTICATION Ldap1 SET host='ldaps://abc.dc.com', binddn_prefix='CN=', binddn_suffix=',OU=Unit2,DC=dc,DC=com', basedn='dc=DC,dc=com', tls_cacert='/home/dc.com.ca.cer', tls_reqcert='never';
```

**Configuring Multiple LDAP Servers**

If you need to configure multiple LDAP servers that have different URLs, create a separate authentication record for each server. Use the PRIORITY keyword to indicate which search the LDAP server performs first.

The following statements create two authentication methods, vldap1 and vldap2. They specify that the LDAP server first search the entire directory (basedn=dc=example,dc=com) for a DN with an OU attribute Sales. If the first search returns no results, or otherwise fails, the LDAP server next searches for a DN with the OU attribute Marketing:

```sql
=> CREATE AUTHENTICATION vldap1 method "ldap" HOST 10.0.0.0/8;
=> ALTER AUTHENTICATION vldap1 SET
    host='ldap://ldap.example.com/search',
    basedn='dc=example,dc=com',
    search_attribute='Sales'
    PRIORITY 1;
=> GRANT AUTHENTICATION vldap1 to public;
=> CREATE AUTHENTICATION vldap2 method "ldap" HOST 10.0.0.0/8;
=> ALTER AUTHENTICATION vldap2 SET
    host='ldap://ldap.example.com/search',
    basedn='dc=example,dc=com',
    search_attribute='Marketing'
    PRIORITY 0;
=> GRANT AUTHENTICATION vldap2 to public;
```

**LDAP Bind Methods**

There are two LDAP methods that you use to authenticate your Vertica database against an LDAP server.

- Bind—Use LDAP bind when Vertica connects to the LDAP server and binds using the CN and password. (These values are the username and password of the user logging into the
database). Use the bind method when your LDAP account's CN field matches that of the username defined in your database. For more information see Workflow for Configuring LDAP Bind.

- Search and Bind —Use LDAP search and bind when your LDAP account's CN field is a user's full name or does not match the username defined in your database. For search and bind, the username is usually in another field such as UID or sAMAccountName in a standard Active Directory environment. Search and bind requires your organization's Active Directory information. This information allows Vertica to log into the LDAP server and search for the specified field. For more information see Workflow for Configuring LDAP Search and Bind.

If you are using search and bind, having a service account simplifies your server side configuration. In addition, you do not need to store your Active Directory password.

**LDAP Anonymous Binding**

*Anonymous binding* is an LDAP server function. Anonymous binding allows a client to connect and search the directory (bind and search) without logging in because binddn and bindpasswd are not needed.

You also do not need to log in when you configure LDAP authentication using Management Console.

**Workflow for Configuring LDAP Bind**

To configure your Vertica database to authenticate clients using LDAP bind, follow these steps:

1. Obtain a service account. For information see the LDAP product documentation. You cannot use the service account in the connection parameters for LDAP bind.

2. Compare the user's LDAP account name to their Vertica username. For example, if John Smith's Active Directory (AD) sAMAccountName = jsmith, his Vertica username must also be jsmith.

   However, the LDAP account does not have to match the database user name, as shown in the following example:
CREATE USER r1 IDENTIFIED BY 'password';
CREATE AUTHENTICATION ldap1 METHOD 'ldap' HOST '172.16.65.177';
ALTER AUTHENTICATION ldap1 SET HOST='ldap://172.16.65.10',basedn='dc=dc,dc=com',binddn_suffix=',ou=unit2,dc=dc,dc=com',binddn_prefix='cn=use';
GRANT AUTHENTICATION ldap1 TO r1;

${TARGET}/bin/vsql -p $PGPORT -U r1 -w $LDAP_USER_PASSWD -h ${HOSTNAME} -c "select user_name, client_authentication_name from sessions;"

3. Run ldapsearch from a Vertica node against your LDAP or AD server. Verify the connection to the server and identify the values of relevant fields. Running ldapsearch helps you build the client authentication string needed to configure LDAP authentication.

In the following example, ldapsearch returns the CN, DN, and sAMAccountName fields (if they exist) for any user whose CN contains the username jsmith. This search succeeds only for LDAP servers that allow anonymous binding:

$ ldapsearch -x -h 10.10.10.10 -b "ou=Vertica Users,dc=CompanyCorp,dc=com" '(cn=jsmith*)' cn dn uid sAMAccountName

ldapsearch returns the following results. The relevant information for LDAP bind is in bold:

# extended LDIF
#
# LDAPv3
# base <ou=Vertica Users,dc=CompanyCorp,dc=com> with scope subtree
# filter: (cn=jsmith*)
# requesting: cn dn uid sAMAccountName
#
# jsmith, Users, CompanyCorp.com
dn: cn=jsmith,ou=Vertica Users,dc=CompanyCorp,dc=com
cn: jsmith
uid: jsmith
# search result
search: 2
result: 0 Success
# numResponses: 2
# numEntries: 1

4. Create a new authentication record based on the information from ldapsearch. In the ldapsearch entry, the CN is username jsmith, so you do not need to set it. Vertica automatically sets the CN to the username of the user who is trying to connect. Vertica uses that CN to bind against the LDAP server.
For more information see [LDAP Bind Parameters](#)

**Workflow for Configuring LDAP Search and Bind**

To configure your Vertica database to authenticate clients using LDAP search and bind, follow these steps:

1. Obtain a service account. For information see the [LDAP product documentation](#).

2. From a Vertica node, run `ldapsearch` against your LDAP or AD server. Verify the connection to the server, and identify the values of relevant fields. Running `ldapsearch` helps you build the client authentication string needed to configure LDAP authentication.

   In the following example, `ldapsearch` returns the CN, DN, and sAMAccountName fields (if they exist) for any user whose CN contains the username, John. This search succeeds only for LDAP servers that allow anonymous binding:

   ```
   $ ldapsearch -x -h 10.10.10.10 -b 'OU=Vertica Users,DC=CompanyCorp,DC=com' -s sub -D 'CompanyCorp\jsmith' -W '(cn=John*)' cn dn uid sAMAccountName
   ```

3. Review the results that `ldapsearch` returns. The relevant information for search and bind is in bold:

   ```
   # extended LDIF
   #
   # LDAPv3
   # base <OU=Vertica Users,DC=CompanyCorp,DC=com> with scope subtree
   # filter: (cn=John*)
   # requesting: cn dn sAMAccountName
   #
   # John Smith, Vertica Users, CompanyCorp.com
dn: CN=jsmith,OU=Vertica Users,DC=CompanyCorp,DC=com
   cn: Jsmith
   sAMAccountName: jsmith
   ```
4. Create the client authentication record. The `cn` attribute contains the username you want—`jsmith`. Set your search attribute to the `CN` field so that the search finds the appropriate account.

```sql
=> CREATE AUTHENTICATION v_ldap_bind_search METHOD 'ldap' HOST '10.10.10.10';
=> GRANT AUTHENTICATION v_ldap_bind_search TO public;
=> ALTER AUTHENTICATION v_ldap_bind_search SET
    host='ldap://10.10.10.10',
    basedn='OU=Vertica,DC=CompanyCorp,DC=com',
    binddn='CN=jsmith,OU=Vertica Users,DC=CompanyCorp,DC=com',
    bind_password='password',
    search_attribute='CN';
```

For more information see [LDAP Search and Bind Parameters](https://Documentation.Vertica.com/Security_and_Authentication/Vertica_Analytic_Database_(9.0.x)/Page_4061_of_6180).
TLS/SSL Server Authentication

The terms SSL and TLS are often used interchangeably. This document uses both terms. TLS is the successor to SSL and offers greater security. The original SSL standard was renamed TLS at the time it became open source. The introduction of TLS began with version 1, which is essentially equal to SSL 3. You use openssl commands to create certificates and keys and TLS syntax to create an authentication method.

SSL Authentication

To protect privacy and verify data integrity, you can configure Vertica and database clients to use Secure Socket Layer (SSL). SSL allows secure connection and communication between the client and the server. The SSL protocol uses a trusted third party called a Certificate Authority (CA). Both the owner of a certificate and the party that relies on the certificate trust the CA.

Vertica supports the following authentication methods under Transport Layer Security (TLS) v1.0, v1.1, and v1.2 protocol:

- **SSL server authentication** — Lets the client confirm the server's identity. The client verifies that the server's certificate and public key are valid and were issued by a certificate authority (CA) listed in the client's list of trusted CAs. This authentication helps prevent man-in-the-middle attacks. See "Prerequisites for SSL Server Authentication and SSL Encryption" in SSL Overview and Configuring SSL.

- **SSL client authentication** — (Optional) Lets the server confirm the client's identity. The server verifies that the client's certificate and public key are valid and were issued by a certificate authority (CA) listed in the server's list of trusted CAs. Client authentication is optional because Vertica can authenticate the client at the application protocol level through user name and password credentials. See "Optional Prerequisites for SSL Server and Client Mutual Authentication" in SSL Overview.

- **Encryption** — Encrypts data sent between the client and database server. This method significantly reduces the likelihood that the data can be read if the connection between the client and server is compromised. Encryption works at both ends of a transaction, regardless of whether SSL Client Authentication is enabled. See "Prerequisites for SSL Server Authentication and SSL encryption" in SSL Overview and Configuring SSL.

- **Data integrity** — Verifies that data sent between the client and server has not been altered during transmission.
Note: For server authentication, Vertica supports using RSA encryption with ephemeral Diffie-Hellman (DH). DH is the key agreement protocol.

Certificate Authority

The CA issues electronic certificates to identify one or both ends of a transaction. These certificates to verify ownership of a public key by the name on the certificate.

Public and Private Keys

A CA issues digital certificates that contain a public key and the identity of the owner. The public key is available to all users through a publicly accessible directory. However, private keys are confidential to their respective owners. When you use a private/public key pair, the data is encrypted by one key and decrypted by its corresponding key.

- If encrypted with a public key, data can be decrypted by its corresponding private key only.
- If encrypted with a private key, data can be decrypted by its corresponding public key only.

For example, suppose Alice wants to send confidential data to Bob. Because she wants only Bob to read it, she encrypts the data with Bob's public key. Even if someone else gains access to the encrypted data, it remains protected. Because only Bob has access to his corresponding private key, he is the only person who can decrypt Alice's encrypted data back into its original form.

SSL Overview

Before you implement SSL security, including mutual mode, obtain the appropriate certificate signed by a certificate authority (CA) and private key files. Copy the certificate file to the database catalog directory. These files must use the Privacy-Enhanced Mail (PEM) format. PEM is the standard file format for Certificates and can be included in ascii or rich text documents. For reference information on SSL use the following links:

- OpenSSL Documentation
- OpenSSL Support
**OpenSSL FAQs**

**OpenSSL Release**

Use the following files for SSL authentication:

- **root.crt** - contains the top-level Certificate Authorities that are trusted for signing server certificates (server.crt).

- **server.crt** - must reside in the server’s data directory and contains the trusted server certificate. This file gets sent to the client where root.crt identifies the server.

- **server.key** - must reside in the server’s data directory and proves the server certificate was sent by the certificate owner. It does not indicate the certificate owner is trustworthy.

If you make changes to any of these files you must restart the server.

You can also implement SSL with LDAP authentication. For more information see [Using LDAP Over SSL/TLS](#).

**Using wildcards**

You can enter wildcard characters as part of the server names in root.crt. For example, server.crt contains server names called eng001.corptech.com, eng002.corptech.com, and eng003.corptech.com. You can enter *.corptech.com in root.crt and it locates the required server.

The wildcard must be the first character of the hostname followed by a period, for example:

```
*.hostname.com
```

**Set Up SSL Server Authentication and SSL Encryption**

Follow these steps to set up server SSL authentication:

**Important:** If you do not perform these steps, database operation may be compromised. If the client cannot authenticate the server, the database does not start.

1. Enable SSL authentication in one of the following ways:

   ```
   Set EnableSSL=1 in vertica.conf
   Enter ALTER DATABASE mydb SET EnableSSL = 1; in vsql.
   ```
2. Copy the server certificate file (server.crt) and private key (server.key) to one of your server hosts in the cluster, as follows:

3. Distribute these files to all server hosts using the instruction in Distributing Certificates and Keys.

   The public key contained in the certificate and the corresponding private key allow the SSL connection to encrypt the data to protect data integrity

4. SSL Server mode requires that the client verify the server's certificate. The client must be able to access certificate authority file (root.crt), and the server must be set with server certificate and private key. Also, SSL Mutual mode requires that the server verify the client’s certificate. The server should be set with CA file and the client must have the client certificate and its private key.

   For vsqI:

   If the VSQL_HOME environment variable is not set, copy the root.crt file to the .vsqI subdir of the login user’s home directory (for example, ~/.vsqI/root.crt).

   If the VSQL_HOME environment variable is set, copy the root.crt file to the .vsqI subdir of the target directory (for example, $vsql_home/.vsqI/root.crt)

   The root.crt file contains the Certificate Authority that issued the server certificate.

Set SSL Server for Mutual Mode Authentication

Use SSL Mutual Mode to have both server and client mutually authenticate themselves with SSL keys. With SSL Mutual Mode the server requests a certificate from the client and the client requests a certificate from the server. Set up SSL Mutual Mode as follows:

1. Enable SSL authentication in one of the following ways:

   Set EnableSSL=1 in vertica.conf

   Enter ALTER DATABASE mydb SET EnableSSL = 1; in vsqI.
2. Copy the root.crt file to one server host in the cluster. This file is distributed to all server hosts when you distribute certificates and keys. See Distributing Certificates and Keys.

   The root.crt file has the same name on the client and server though the file contents can differ. The contents are identical only if the client and server certificates were used by the same root certificate authority (CA).

3. Copy the client certificate file (client.crt) and private key (client.key) to each client. For vsql:

   - If the VSQL_HOME environment variable is set, copy the file to the .vsql subdirectory of the target directory set up in the environment variable (e.g., $vsql_home/.vsql/client.crt).

   - If the VSQL_HOME environment variable is not set, copy the two files to the .vsql subdir of the login user's home directory. (e.g., ~/.vsql/client.crt).

   If you are using either ODBC or JDBC, you can place the files anywhere on your system. Then, provide the location in the connection string (ODBC/JDBC) or ODBCINI (ODBC only). See Configuring SSL for ODBC Clients and Configuring SSL for JDBC Clients.

   Important: If you're using ODBC, the private key file (client.key) must have read and write permissions only for the dbadmin user. For example:

   ```
   chmod 600 client.key
   ```

   Do not provide any additional permissions or extend them to any other users.

### Generating SSL Certificates and Keys

Generating SSL certificates and keys, you must perform the following tasks:

- **Create a Certificate Authority Private Key and Public Certificate** that can then be used to sign server and client keys.

  Important: In a production environment, always use certificates signed by a Certificate Authority.

- **Create the Server Private key and Certificate**, and request a new server certificate that includes a public key.
Create the Client Private key and Certification, and request a new client certificate that includes a public key.

For more detailed information on creating signed certificates, refer to the OpenSSL documentation.

The documentation includes examples and sample procedures to show how to create certificates and keys. The commands shown allow many other possible options not used in these examples. Create commands based on your specific environment.

Create a Certificate Authority Private Key and Public Certificate

Create a Certificate Authority (CA) private key and public certificate. For more information on using the following commands. see the OpenSSL documentation.

1. Generate CA files serverca.crt and servercakey.pem. This allows the signing of server and client keys:

```
$ openssl genrsa -out new_servercakey.pem
$ openssl req -config openssl_req_CA.conf -new -x509 -key servercakey.pem -out serverca.crt
```

You can add multiple CA files to the .crt file with the following command:

```
$ cat serverca_new.crt >> serverca.crt
```

This concatenates the serverca_new.crt file to the original CA file serverca.crt. You can run this command multiple times to concatenate additional CA files.

2. Create the server private key (server.crt) and public key (server.key):

```
$ openssl genrsa -out server.key
$ openssl req -config openssl_req_server.conf -new -key server.key -out server_reqout.txt
$ openssl x509 -req -in server_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt -CAkey servercakey.pem -out server.crt
```

3. Create the client private key (client.crt) and public key (client.key):

```
$ openssl genrsa -out client.key
$ openssl req -config openssl_req_client.conf -new -key client.key -out client_reqout.txt
$ openssl x509 -req -in client_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt -CAkey servercakey.pem -out client.crt
```
4. Enter the following sample CA certificate values in response to openssl command line prompts. The actual values you enter here will be different than the sample values.

Rather than enter these values from command line prompts, you can optionally provide the same information in .conf files. For example, openssl_req_ca.conf in the preceding example.

- Country Name (2 letter code) [GB]: US
- State or Province Name (full name) [Berkshire]: Massachusetts
- Locality Name (e.g., city) [Newbury]: Cambridge
- Organization Name (e.g., company) [My Company Ltd]: CorpName
- Organizational Unit Name (e.g., section) []: TechSupport
- Common Name (e.g., your name or server hostname) []: myhost
- Email Address []: myhost@CorpName.com

5. Include one unique Distinguished Name (DN) for each certificate that you create. In the preceding examples, the DN is the Organizational Unit Name.

6. Set file permissions:

   $ chmod 700 server.crt server.key
   $ chmod 700 client.crt client.key

7. Rename the CA file serverca.crt to root.crt, and do one of the following:

   - copy it to VSQL_HOME or
   - point to it in ODBC DSN Configuration dialog

8. Create a trusted Certificate Authority for the client for server mutual mode:

   => ALTER DATABASE <mydb> SET SSLCA = '\content of root.crt';

You now have a CA private key, new_servercakey.pem. You also have a CA public certificate, new_serverca.crt. Use both the private key and the public certificate in the procedures that follow for creating server and client certificates.

**Generating Certificates and Keys for MC**

A certificate signing request (CSR) is a block of encrypted text generated on the server on which the certificate is used. You send the CSR to a certificate authority (CA) to apply for a digital identity certificate. The CA uses the CSR to create your SSL certificate from information in your certificate; for example, organization name, common (domain) name, city, and country.
Management Console (MC) uses a combination of OAuth (Open Authorization), Secure Socket Layer (SSL), and locally-encrypted passwords to secure HTTPS requests between a user's browser and MC, and between MC and the agents. Authentication occurs through MC and between agents within the cluster. Agents also authenticate and authorize jobs.

The MC configuration process sets up SSL automatically, but you must have the openssl package installed on your Linux environment first.

When you connect to MC through a client browser, Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your Vertica cluster, and on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See Set Up Time Synchronization in Installing Vertica.

Create a Certificate and Submit it for Signing

For production, you must use certificates signed by a certificate authority. You can create and submit a certificate and when the certificate returns from the CA, import the certificate into MC.

Use the openssl command to generate a new CSR:

```
$ sudo openssl req -new -key /opt/vconsole/config/keystore.key -out server.csr
```

When you press Enter, you are prompted to enter information to be incorporated into your certificate request. Some fields contain a default value, which you should change for security reasons. Other fields you can leave blank, such as password and optional company name. To leave the field blank, type ' . '.

Important: The keystore.key value for the -key option creates private key for the keystore. If you generate a new key and import it using the Management Console interface, the MC process does restart properly. You must restore the original keystore.jks file and restart Management Console.

This information is contained in the CSR and shows both the default and replacement values:

```
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [GB]:Massachusetts
Locality Name (eg, city) [Newbury]: Cambridge
Organization Name (eg, company) [My Company Ltd]:Vertica
Organizational Unit Name (eg, section) []:Information Management
Common Name (eg, your name or your server's hostname) []:console.vertica.com
Email Address []:mcadmin@vertica.com
```
The Common Name field is the fully qualified domain name of your server. Your entry must exactly match what you type in your web browser, or you receive a name mismatch error.

Self-Sign a Certificate for Testing

To test your new SSL implementation, you can self-sign a CSR using either a temporary certificate or your own internal CA, if one is available.

Note: A self-signed certificate generates a browser-based error notifying you that the signing certificate authority is unknown and not trusted. For testing purposes, accept the risks and continue.

The following command generates a temporary certificate, which expires after 365 days:

```
$ sudo openssl x509 -req -days 365 -in server.csr -signkey /opt/vconsole/config/keystore.key -out server.crt
Enter passphrase for /opt/vconsole/config/keystore.key:
Enter same passphrase again:
```

The previous example prompts you for a passphrase. This is required for Apache to start. To implement a passphrase you must put the SSLPassPhraseDialog directive in the appropriate Apache configuration file. For more information see your Apache documentation.

This example shows the command's output to the terminal window:

```
Signature ok
subject=/C=US/ST=Massachusetts/L=Cambridge/O=Vertica/OU=IT/CN=console.vertica.com/emailAddress=mcadmin@vertica.com
Getting Private key
```

You can now import the self-signed key, server.crt, into Management Console.

See Also

- Configuring SSL
- Key and Certificate Management Tool

Importing a New Certificate to MC

Use this procedure to import a new certificate into Management Console.

Note: To generate a new certificate for Management Console, you must use the keystore.key file, which is located in /opt/vconsole/config on the server on which
you installed MC. Any other generated key/certificate pair causes MC to restart incorrectly. You will then have to restore the original keystore.jks file and restart Management Console. See Generating Certifications and Keys for Management Console.

1. Connect to Management Console, and log in as an administrator.

2. On the Home page, click MC Settings.

3. In the button panel on the left, click SSL certificates.

4. To the right of "Upload a new SSL certificate," click Browse to import the new key.

5. Click Apply.

6. Restart Management Console.

Configuring SSL

Configure SSL for each server in the cluster.

1. Verify that you have performed at least the minimum steps required in SSL Overview for server authentication and encryption and, optionally, for mutual authentication.

2. Verify that you have performed the steps in Distributing Certificates and Keys.

Important: Before you set the Security Parameters SSLCertificate and SSLPrivateKey, you must first set the EnableSSL parameter. Admintools sets these parameters for you when you perform the procedure steps listed in Distributing Certificates and Keys. Alternatively, you can use vsq1 to set the parameters using the ALTER DATABASE statement. For more information on setting configuration parameters see ALTER DATABASE.

These parameters are also automatically set during upgrade to 7.1 if you set EnableSSL=1 in the previous version.

3. Set the EnableSSL parameter to 1. By default, EnableSSL is set to 0 (disabled).

   => ALTER DATABASE mydb SET EnableSSL = 1;

4. Restart the database.

5. If you are using either ODBC or JDBC, configure SSL for the appropriate client:
Configuring SSL for ODBC Clients

Configuring SSL for JDBC Clients

vsql automatically tries to connect using SSL. If a connection fails, and your server is started in SSL Server Mode, vsql attempts to make a second connection over clear text. If you start the server in SSL Mutual Mode, the connection fails without vsql attempting to connect over clear text.

Configure JDBC for SSL Mutual Mode

In addition to the procedure above, you need to perform the following to configure SSL in Mutual Mode:

a. Edit openssl_req_server.conf as follows:

```
[req]
prompt = no
distinguished_name = CStore4Ever
req_extensions = v3_req

[CStore4Ever]
C = US
ST = Massachusetts
O = Corp Server
CN = engXXX
e-mailAddress = foo@bar.com

[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = engXXX.corp.com
DNS.2 = engXXX
DNS.3 = *.corp.com
IP = 10.20.40.XX
```

b. Create the server private key (server.crt) and public key (server.key):

```
$ openssl genrsa -out server.key
$ openssl req -config openssl_req_server.conf -new -key server.key -out server_reqout.txt
$ openssl x509 -req -in server_reqout.txt -days 3650 -sha1 -CAcreateserial -CA serverca.crt
-CAkey servercakey.pem -extensions v3_req -extfile openssl_req_server.conf -out server.crt
```

Note: If you are using SSL mutual mode with JDBC, copy the root.crt file to a location on any one of the clients. After you copy the file to the client, root.crt is
Configuring SSL for ODBC Clients

Configuring SSL for ODBC clients requires that you set the SSLMode connection property. If you want to configure optional SSL client authentication, you must also configure the Security Parameters SSLKeyFile and SSLCertFile connection properties.

How you configure the DSN depends on your operating system:

- **Linux and UNIX** — Enter the connection properties in the odbc.ini file. See [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX Clients](#).

- **Microsoft Windows** — Enter the connection properties in the Windows Registry. See [Creating an ODBC DSN for Windows Clients](#).

For Windows ODBC you can set connection string properties in the Client Settings tab on the ODBC DSN Configuration dialog. Connection string properties you can set are:

- **SSL CA file** - the location of the root certificate file, for example root.crt.

- **SSL cert file** - the location of the server certificate file, for example server.crt.

- **SSL key file** - the location of the server key file, for example, server.key.

Set SSLMode Connection Property

Set the SSLMode connection property to one of the following options for the DSN:

- Prefer
- Require
- Disable
- Auto
- VerifyOptional
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verify_full</td>
<td>Encrypts data and connects to a user-specified trusted server.</td>
</tr>
<tr>
<td>verify_ca</td>
<td>Encrypts data and connects to a trusted server.</td>
</tr>
<tr>
<td>require</td>
<td>Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails.</td>
</tr>
<tr>
<td>prefer</td>
<td>(Default value) Indicates your preference that the server to use SSL. The first connection to the database tries to use SSL. If that connection fails, a second connection is attempted over a clear channel.</td>
</tr>
<tr>
<td>allow</td>
<td>Makes a connection to the server whether the server uses SSL or not. The first connection attempt to the database is made over a clear channel. If that connection fails, a second connection is attempted over SSL.</td>
</tr>
<tr>
<td>disable</td>
<td>Never connects to the server using SSL. This setting is typically used for troubleshooting.</td>
</tr>
</tbody>
</table>

Using verify_ca and verify_full

You can use the SSL Mode properties verify_ca and verify_full exclusively on client authentication. These properties require:

- The root.crt to allow the client to access the server’s authentication certificate. If root.crt does not exist, and verify_can or verify_full are set, the connection fails upon SSL initialization.

- A server.crt, which contains the authentication certificate.

- A server.key, which contains the server’s private key that prevents the server from accessing external systems.

- A server hostname that matches the hostname specified by the client, verify_full only.

- A certificateAll nodes in the cluster must either share the same certificate or each certificate must contain all the hostnames or IP addresses in the Subject Alternative Name (SAN).

SSL Workflow

The following diagram shows an example workflow for SSL authentication. Your actual workflow may differ depending on what SSLMode Connection Properties you use.
In this workflow:

1. **SSL Mode**
   - **SSLMode Connection Props prefer, require, verify_ca or verify_full**
2. **Has root.crt?**
   - **No**
     - **SSL Mode?**
       - **verify_ca or verify_full**
         - **Connection fails**
   - **prefer or require**
     - **SSL Request**
       - **Assuming SSL enabled**
         - **Get server certificate and public key**
           - **succeed**
             - **SSL Authentication**
               - **succeed**
                 - **Connection succeeds and session starts**
               - **fail**
                 - **Connection fails**
         - **fail**
           - **Connection fails**
         - **SSL Mode?**
           - **verify_full**
             - **Hostname check**
               - **fail**
                 - **Connection fails**
               - **succeed**
                 - **Connection succeeds and session starts**
If the SSLMode Connection Property is set to none or allow, the client connects without authentication.

If the SSLMode Connection Property is set to verify_ca or verify_full, and root.crt does not exist, the SSL authentication fails. If root.crt exists, the authentication process proceeds.

During SSL authentication if the SSLMode Connection Property is set to verify_full, and the server hostname is not the same as the hostname specified by the client, authentication fails.

Enable SSL Mutual Mode Authentication

You can optionally configure SSL mutual mode by setting the following database Security Parameters:

- SSLKeyFile — Set this connection property to the file path and name of the client's private key. This key can reside anywhere on the client.

- SSLCertFile — Set this connection property to the file path and name of the client's public certificate. This file can reside anywhere on the client.

Configuring SSL for JDBC Clients

Configuring SSL Authentication for JDBC clients involves the following tasks:

- Set required properties
- Optionally run the SSL Debug Utility
- Configure for SSL Mutual Mode (optional)

Set Required Properties

Set Properties When Location or the Keystore/Truststore Is Not the Default

If you are using a location or the keystore/truststore that is not the default, set the following system properties so that the JRE can find your keystore/truststore:

```
$ javax.net.ssl.keyStore
$ javax.net.ssl.trustStore
```
Set Properties When Keystore/Truststore Is Password Protected

If your keystore/truststore is password protected, set the following system properties so that the JRE has access to it:

```
$ javax.net.ssl.keyStorePassword
$ javax.net.ssl.trustStorePassword
```

Run the SSL Debug Utility

After configuring SSL for JDBC, optionally run the following command to enable the debug utility for SSL:

```
$ -Djavax.net.debug=ssl
```

You can use several debug specifiers (options) with the debug utility. The specifiers help narrow the scope of the debugging information that is returned. For example, you could specify one of the options that prints handshake messages or session activity.

For information on the debug utility and its options, see Debugging Utilities in the Oracle document, JSSE Reference Guide.

For information on interpreting debug information, refer to the Oracle document, Debugging SSL/TLS Connections.

Distributing Certificates and Keys

Before you can distribute certifications and keys to all hosts in a cluster, you must obtain the appropriate certificate signed by a certificate authority (CA) and private key files. See SSL Overview.

To distribute certifications and keys to all hosts in a cluster:

1. Log on to a host that contains the certifications and keys you want to distribute.

2. Start the Administration Tools, as described in Using the Administration Tools

   Note: The database does not need to be running when you distribute the certificates and key files.
3. On the Main Menu in the Administration Tools, select Configuration Menu, and click OK.

4. On the Configuration Menu:
   a. Select Distribute Config Files, and click OK.
   
   b. Select SSL Keys, and click OK.

5. Select the database on which you want to distribute the files and click OK. The following appears:

![Select files to install](image)

6. [Optional] Modify the fields in the previous screenshot to add the file locations for the `server.crt`, `server.key` and `root.crt` files, and click OK to distribute the files.

   Admintools sets the parameters SSLCertificate, SSLPrivateKey, and, if applicable, SSLCA. See Security Parameters.

   - If you are upgrading to 7.1, the SSLCertificate and SSLPrivateKey parameters are automatically set by Admintools if you set EnableSSL=1 in the previous version.
   - If your server.crt SSL certificate file includes certificate chain (more than one certificate), Admintools accepts the whole chained certificate.

7. Configure SSL.
**TLS Authentication**

Server authentication methods define how clients connect to a Vertica server. Before you define a TLS authentication method, you should understand what type of authentication methods your Vertica server supports. You should also perform any prerequisite tasks.

In regards to SSL, your server can operate with:

- No SSL
- SSL Server Mode —The client does not need certificate or key files.
- SSL Mutual Mode —The client needs certificate, key, and certificate authority files.

SSL modes are independent of authentication, except that the SSL client self-authentication method requires that your server be set-up in SSL Mutual Mode. Otherwise, if you are not implementing client self-authentication method, you can use TLS authentication with either SSL Server Mode or SSL Mutual Mode.

Before you create a TLS authentication method, perform the pre-requisite tasks necessary for your specific environment (for example, certificate creation). Refer to TLS/SSL Server Authentication and all subsections applicable to your environment.

To create a TLS authentication method, use the command `CREATE AUTHENTICATION` as documented in the SQL Reference Manual.

**Implementing Client Self-Authentication**

To use a client self-authentication method, your server must be in SSL Mutual Mode.

To create an authentication method for client self-authentication, use the `CREATE AUTHENTICATION` statement. Specify the `auth_type 'tls'` and with HOST TLS.

**Important:** You use the `tls` `auth_type` only when you want to create an authentication method for client self-authentication. You must use the `tls` `auth_type` with the HOST TLS syntax.
Create an Authentication Method with Client Self-Authentication Method

This section provides sample chronological steps for setting up a client for self-authentication, creating an authentication method, and associating the method with a user through a grant statement.

1. Follow all applicable procedures for implementing SSL and distributing certificates and keys. Refer to TLS/SSL Server Authentication as it applies to your environment.

   When you create a client key, make sure to include a Common Name (CN) that is the database user name you want to use with the target database.

   $ Common Name <server hostname> []:<database username>

2. Create the authentication method. Authentication methods are automatically enabled when you create them.

   => CREATE AUTHENTICATION myssltest METHOD 'tls' HOST TLS '10.0.0.0/23;

3. Associate the method with the user through a grant statement.

   => GRANT AUTHENTICATION myssltest TO mydatabaseusername;

Your client can now log on and be recognized.

For information on creating authentication methods, refer to the SQL Reference Manual topic, CREATE AUTHENTICATION.

Specify TLS for Client Connections

You can require clients to use TLS when connecting to Vertica. To do so, create a client authentication method for them that uses the HOST TLS syntax with the CREATE AUTHENTICATION statement.

Specific clients might connect through a network connection known to be insecure. In such cases, you can choose to limit specific users to connecting through TLS. You can also require all clients to use TLS.

See Creating Authentication Records for more information about creating client authentication methods.
LDAP Link Service

LDAP Link enables synchronization between the LDAP and Vertica servers. This eliminates the need for you to manage two sets of users and groups or roles, one on the LDAP server and another on the Vertica server. With LDAP synchronization, the Vertica server becomes a replication database for the LDAP server.

Automatic Synchronization

With LDAP Link the Vertica server closely integrates with an existing directory service such as MS Active Directory or OpenLDAP. The Vertica server automatically synchronizes:

- LDAP users to Vertica users
- LDAP groups to Vertica roles

You manage all user and group properties in the LDAP server. If you are the Vertica database administrator, you need only to set up permissions for Vertica Analytic Database access on the users and groups.

Configure LDAP Link with LDAP Link connection parameters that reside in the catalog. See Set LDAP Link Parameters for more information.

Enable LDAP Link

Enable LDAP Link as shown:

```sql
=> ALTER DATABASE dbname SET PARAMETER LDAPLinkURL='ldap://example.dc.com',
LDAPLinkSearchBase='dc=DC,dc=com', LDAPLinkBindDN='CN=jsmith,OU=QA,DC=dc,DC=com,
LDAPLinkBindPswd='password',LDAPLinkFilterUser='(objectClass=inetOrgPerson)',
LDAPLinkFilterGroup='(objectClass=group)', LDAPLinkOn=1;
=> SELECT ldap_link_sync_start();
```

See LDAP Link Parameters.

LDAP Link Workflow

After you enable LDAP Link, synchronization occurs according to this workflow:
1. The System Administrator creates users and user groups on the LDAP server.

2. The System Administrator sets up LDAP Link service parameters as required and enables the service.

3. Using the LDAP Link service, Vertica Analytic Database replicates the users and user groups from the Application LDAP to the Vertica database, creating Vertica users and roles.

4. The LDAP server uses Kerberos (KDC) to authenticate the user logging in to Vertica.
   - The LDAP user can log into Vertica if assigned the appropriate authentication type.
   - After login, you can grant users privileges using GRANT statements or as part of a Group.

Note: After synchronization the Vertica Analytic Database user does not have an associated authentication method. To allow the user to login, you must assign an authentication method to the user. See Implementing Client Authentication.
Using LDAP Link

When you implement LDAP Link the following are directly affected and help you manage and monitor the LDAP Link - Vertica Analytic Database synchronization:

- User and Group management
- LDAP Link User Flag
- Blocked Commands
- Client Authentication types

User and Group Management

Users and groups created on the LDAP server have a specific relationship with those users and roles replicated to the Vertica server:

- The user-group relationship on the LDAP server is maintained when those users and groups (roles) are synchronized with Vertica Analytic Database.

- If a user or group name exists on the Vertica database and a user or group with the same names is synchronized from the LDAP Server using LDAP Link, the users or groups become conflicted. Vertica can not support multiple users with the same name. For a resolution of this conflict, see User Conflicts.

LDAP Link uses the entries in the dn: section of the LDAP configuration file as the unique user identifier when synchronizing a user to the Vertica Analytic Database:

```
dn: cn=user1,ou=dev,dc=example,dc=com
  cn: user1
  ou: dev
  id: user1
```

The uid parameter in the LDAP configuration file indicates the LDAP user name.

```
uid: user1
```

Upon synchronization, the dn: entry gets mapped to the uid: to identify the Vertica Analytic Database user.
If you change a setting in the dn: and do not change the uid: LDAP Link interprets the user as a new user when re-synchronizing with the Vertica Analytic Database. In this case, the existing Vertica Analytic Database user with that uid: gets deleted from Vertica and a new Vertica Analytic Database user is created.

If you change the uid: and not the dn: on LDAP, the uid on the Vertica Analytic Database gets updated to the new uid. Since you did not change the dn: LDAP Link does not interpret the user as a new user.

### LDAP Link User Flag

As a dbadmin user, you can access the vs_users table to monitor user behavior on the Vertica Analytic Database. The users table contains an `ldap_dn` field that identifies whether or not the Vertica Analytic Database user is also an LDAP Link user. This example shows the `ldap_dn` field set to `dn` indicating the Vertica Analytic Database user is also an LDAP Link user:

```sql
=> SELECT * FROM vs_users;
- [ RECORD 1 ]-+-------------------------------+
| user_id       | 45035996273704962 |
| user_name     | dbadmin           |
| is_superuser  | t                 |
| profile_name  | default           |
| is_locked     | f                 |
| lock_time     |                  |
| resource_pool | general           |
| memory_cap_kb | unlimited         |
| temp_space_cap_kb | unlimited    |
| run_time_cap  | unlimited         |
| max_connections| unlimited        |
| connection_limit_mode | database     |
| idle_session_timeout | unlimited |
| all_roles     | dbuser*, dbadmin*, pseudosuperuser* |
| default_roles | dbuser*, dbadmin*, pseudosuperuser* |
| search_path   |                  |
| ldap_dn       | dn                |
| ldap_uri_hash | Ø                 |
| is_orphaned_from_ldap | f        |
```

### Blocked Commands

Be aware that the following SQL statements are blocked for Vertica Analytic Database users with `ldapdn` set to `dn` in the `vs_users` table:

- **DROP USER** and **DROP ROLE**
- **ALTER ROLE RENAME**
- `ALTER USER` name IDENTIFIED BY 'password' [REPLACE 'old_password']
- `ALTER USER` name PASSWORD EXPIRE
- `ALTER USER` name PROFILE
- `ALTER USER` name SECURITY_ALGORITHM...
- `ALTER USER` name DEFAULT ROLE role-name
- `GRANT (Role)`

### Client Authentication Types

LDAP user and groups cannot log into Vertica if client authentication is not assigned to the user or group. You can use the following valid authentication types for LDAP users and groups:

- GSS
- Ident
- LDAP
- Reject
- Trust

### LDAP Link Parameters

Use LDAP Link parameters to determine:

- LDAP Link operations, such as enabling or disabling LDAP Link and how often to perform replication
- Authentication parameters, including SSL authentication parameters
- Users and groups that inherit unowned objects
- How to resolve conflicts
Set LDAP Link Parameters

This example shows how you can set:

- The URL of the LDAP server (LDAPLinkURL) and
- The base DN from where to start replication (LDAPLinkSearchBase)

You also see how to set the LDAP Link Bind authentication parameters (LDAPLinkBindDN and LDAPLinkBindPswd) and enables LDAP Link (LDAPLinkOn).

```sql
=> ALTER DATABASE myDB1 SET PARAMETER LDAPLinkURL='ldap://10.60.55.128',
   LDAPLinkSearchBase='dc=corp,dc=com',LDAPLinkBindDN='dc=corp,dc=com',LDAPLinkBindPswd='password';

=> ALTER DATABASE myDB1 SET PARAMETER LDAPLinkOn = '1';
```

General and Connection Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAPLinkOn</td>
<td>Enables or disables LDAP Link.</td>
</tr>
<tr>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>LDAP Link disabled</td>
</tr>
<tr>
<td>1</td>
<td>LDAP Link enabled</td>
</tr>
<tr>
<td><strong>Default value:</strong></td>
<td>0</td>
</tr>
<tr>
<td>LDAPLinkURL</td>
<td>The LDAP server URL.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td>SET PARAMETER</td>
<td>LDAPLinkURL='ldap://glw2k8-64.dc.com';</td>
</tr>
<tr>
<td>LDAPLinkInterval</td>
<td>The time interval, in seconds, by which the LDAP Server and</td>
</tr>
<tr>
<td></td>
<td>Vertica server synchronize.</td>
</tr>
<tr>
<td><strong>Default Value:</strong></td>
<td>86400 (one day).</td>
</tr>
<tr>
<td>LDAPLinkFirstInterval</td>
<td>The first interval, in seconds, for LDAP/Vertica</td>
</tr>
<tr>
<td></td>
<td>synchronization after the clerk node joins the cluster.</td>
</tr>
<tr>
<td><strong>Default Value:</strong></td>
<td>120</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LDAPLinkRetryInterval</td>
<td>The time, in seconds, the system waits to retry a failed synchronization.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 10</td>
</tr>
<tr>
<td>LDAPLinkRetryNumber</td>
<td>The number of retry attempts if synchronization failed.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 10.</td>
</tr>
<tr>
<td>LDAPLinkSearchBase</td>
<td>The base dn from where to start replication.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td><code>SET PARAMETER LDAPLinkSearchBase='ou=vertica,dc=mycompany,dc=com';</code></td>
</tr>
<tr>
<td></td>
<td>Vertica recommends using a separate OU for database users.</td>
</tr>
<tr>
<td>LDAPLinkSearchTimeout</td>
<td>The timeout length, in seconds, for the LDAP search operation during an LDAP Link Service run.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 10</td>
</tr>
<tr>
<td>LDAPLinkScope</td>
<td>Indicates what dn level to replicate.</td>
</tr>
<tr>
<td></td>
<td>Valid Values:</td>
</tr>
<tr>
<td></td>
<td>- sub — Replicate entire subtree under baseDN</td>
</tr>
<tr>
<td></td>
<td>- one — Replicate to one level under baseDN</td>
</tr>
<tr>
<td></td>
<td>- base — Replicate only the baseDN level</td>
</tr>
<tr>
<td></td>
<td>If you decrease the scope (for example, sub to one), some users may not be recognized during the next synchronization.</td>
</tr>
<tr>
<td></td>
<td>Default Value: sub</td>
</tr>
<tr>
<td>LDAPLinkFilterUser</td>
<td>Determines how to filter users to be replicated.</td>
</tr>
<tr>
<td></td>
<td>Default Value: &quot;(objectClass=inetOrgPerson)&quot;</td>
</tr>
<tr>
<td>LDAPLinkFilterGroup</td>
<td>Determines how to filter groups to be replicated.</td>
</tr>
<tr>
<td></td>
<td>Default Value: &quot;(objectClass=groupofnames)&quot;</td>
</tr>
<tr>
<td>LDAPLinkGroupName</td>
<td>[Optional] The LDAP field to use when creating a role name in Vertica.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Value: cn</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LDAPLinkGroupMembers</strong></td>
<td>The LDAP group name linked to a user.</td>
</tr>
<tr>
<td><strong>Default Value: member</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LDAPLinkUserName</strong></td>
<td>The LDAP field to use when creating a user name in Vertica.</td>
</tr>
</tbody>
</table>

### Authentication Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LDAPLinkBindDN</strong></td>
<td>The LDAP Bind DN used for authentication.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>SET PARAMETER LDAPLinkBindDN='CN=amir,OU=QA,DC=dc,DC=com';</code></td>
</tr>
<tr>
<td><strong>LDAPLinkBindPswd</strong></td>
<td>The valid password for the LDAP Bind DN to access the server. Only accessible by the dbadmin user.</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>SET PARAMETER LDAPLinkBindPswd='password';</code></td>
</tr>
</tbody>
</table>

### SSL Authentication Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LDAPLinkStartTLS</strong></td>
<td>[Optional] Specifies whether or not to use the StartTLS operation during bind. You can only use this parameter if the LDAP server’s URL is &quot;ldap://...&quot; (not &quot;ldaps://...&quot;)</td>
</tr>
<tr>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td>0 - Do not use starttls</td>
<td></td>
</tr>
<tr>
<td>1 - Use starttls</td>
<td></td>
</tr>
<tr>
<td><strong>Default Value: 0</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LDAPLinkTLSReqCert</strong></td>
<td>[Optional] Specifies how to manage certificates when</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>using TLS.</td>
</tr>
<tr>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td>allow</td>
<td>connection is successful even if the client does not provide a certificate or provides an invalid certificate.</td>
</tr>
<tr>
<td>hard</td>
<td>connection is unsuccessful if the client does not provide a certificate or provides an invalid certificate.</td>
</tr>
<tr>
<td>try</td>
<td>connection is successful if the client does not provide a certificate. A connection is unsuccessful if the client provides an invalid certificate.</td>
</tr>
<tr>
<td>never</td>
<td>The client does not request or verify a certificate. For more information see <a href="#">Using LDAP Over SSL/TLS</a>.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>allow</td>
</tr>
</tbody>
</table>

| LDAPLinkTLSCACert       | [Optional] The path to the CA certificates. |

### Miscellaneous Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAPLinkConflictPolicy</td>
<td>Determines how to resolve a user conflict.</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Values:</strong></td>
</tr>
<tr>
<td>IGNORE</td>
<td>Ignores the incoming LDAP user and maintains the existing Vertica user.</td>
</tr>
<tr>
<td>MERGE</td>
<td>Converts the existing user to an LDAP user.</td>
</tr>
<tr>
<td>Default Value:</td>
<td>MERGE</td>
</tr>
</tbody>
</table>

| LDAPLinkStopIfZeroUsers | Enables or disables the shutdown of LDAPLink synchronization if no users are found in LDAP. |
|                         | **Valid values:** |
| 0                       | Disables the LDAPLink synchronization shutdown if no users are found. This may lead to inadvertent dropping of Vertica users. |
| 1                       | Enables the LDAPLink synchronization shutdown if no |
### Troubleshooting LDAP Link Issues

Various issues can arise with LDAP Link Service, including:

- Disconnected (Orphaned) Users and Roles
- Lost Objects
- User Conflicts

### Disconnected (Orphaned) Users and Roles

Vertica Analytic Database users and roles synchronized through LDAP Link can become disconnected, or *orphaned*, if an issue arises with the LDAP Link service. For example, users and roles become orphaned when you change the connection to the LDAP server as the following scenario describes:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>are found. This prevents inadvertent dropping of Vertica users.</td>
</tr>
</tbody>
</table>
| LDAPLinkDryRun  | [Optional] Tests the connection to the LDAP server and logs the response without doing a synchronization. Also tests if parameters are correctly set.  
|                 | **Valid Values:**  
|                 | 0 - Disables LDAPLinkDryRun  
|                 | 1 - Enables LDAPLinkDryRun  
|                 | **Default Value:** 0                                                      |

See [Managing Configuration Parameters: VSQL](#) for information on setting LDAP Link parameters.

**Note:** When you change any Connection or Authentication parameter, LDAP Link reconnects and re-initializes the synchronization.
1. Create an LDAP connection as follows:

```sql
=> ALTER DATABASE MyDB1 SET PARAMETER LDAPLinkURL='ldap://ebuser',
  LDAPLinkSearchBase='dc=example,dc=com', LDAPLinkBindDN='mega',
  LDAPLinkBindPswd='$megapassword$';
=> ALTER DATABASE MyDB1 SET PARAMETER LDAPLinkOn = '1';
```

2. Run an LDAP Link session to synchronize LDAP and Vertica users.

3. Change one or more connection parameters from Step 1. You can change the connection only if you change one of the LDAPLinkURL or LDAPLinkSearchBase parameters.

4. Run another LDAP Link session. The system attempts to re-synchronize LDAP and Vertica users. Since the connection has changed, the existing Vertica users cannot be synchronized with the LDAP users from the new connection. These Vertica users become orphaned.

As a dbadmin user, you can identify orphaned users by checking the field is_orphaned_from_ldap in the users system table:

```sql
=> SELECT is_orphaned_from_ldap FROM users;
```

A field value of t indicates that the user is an orphaned user. Orphaned Vertica users cannot connect to the LDAP server and cannot login to Vertica using LDAP authentication (however, other authentication methods assigned to the user work). In this case, you can delete the orphaned Vertica user and run the LDAP Link service to resynchronize users.

## Re-parented Objects

When you delete users or groups from linked LDAP, the LDAP Link service removes the same users and roles from Vertica Analytic Database. However, the service does not delete objects owned by the deleted user. Use the `GlobalHeirUserName` parameter to assign the objects to a new owner (re-parent).

**Example:**

```sql
=> ALTER DATABASE example_db SET PARAMETER GlobalHeirUserName=user1;
```

This creates a new user named user1, if it does not exist. The `GlobalHeirUserName` user serves as the new parent for all the objects owned by deleted users.

By default, this parameter is set to `<auto>` which re-parents the objects to the dbadmin user.

If you leave `GlobalHeirUserName` empty, the objects are not re-parented to another user.

For more information see `GlobalHeirUserName` in Security Parameters.
User Conflicts

Vertica Analytic Database users and roles synchronized using LDAP Link can become conflicted. Such conflicts can occur, for example, when you create a new user or group on the LDAP server and another user or role with the same name exists on the Vertica Analytic Database.

As a dbadmin user, use one of the following parameters to resolve user conflicts:

- LDAPLinkConflictPolicy
- LDAPLinkStopIfZeroUsers

**LDAPLinkConflictPolicy**

Use LDAPLinkConflictPolicy to resolve any user conflicts:

- LDAPLinkConflictPolicy=IGNORE - Ignores the incoming LDAP users and maintains the existing Vertica user
- LDAPLinkConflictPolicy=MERGE - Merges the incoming LDAP user with the Vertica user and converts the database user to an LDAP user retaining the database user's objects

Example:

```sql
=> ALTER DATABASE example_db SET PARAMETER LDAPLinkConflictPolicy='MERGE';
```

The default is MERGE. If you change LDAPLinkConflictPolicy, the change takes affect on the next synchronization.

**LDAPLinkStopIfZeroUsers**

Use LDAPLinkStopIfZeroUsers to prevent an accidental dropping of Vertica users if the LDAP Link synchronization does not find any LDAP users.

- LDAPLinkStopIfZeroUsers=0 - Does not stop the LDAP Link synchronization if no users are found in LDAP. This drops all Vertica users during synchronization.
- LDAPLinkStopIfZeroUsers=1 - Stops the LDAP Link synchronization if no users are found in LDAP and displays an error. This prevents the dropping of Vertica users due to some issue.
Monitoring LDAP Link

Use the ldap_link_events table to monitor events that occurred during an LDAP Link synchronization:

```sql
=> SELECT transaction_id, event_type, entry_name, entry_oid FROM ldap_link_events;
```

<table>
<thead>
<tr>
<th>transaction_id</th>
<th>event_type</th>
<th>entry_name</th>
<th>entry_oid</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273705317</td>
<td>SYNC_STARTED</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>4506696273253589</td>
<td>SYNC_FINISHED</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>45066988112255317</td>
<td>PROCESSING_STARTED</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2341123456789765</td>
<td>USER_CREATED</td>
<td>tuser</td>
<td>234548899</td>
</tr>
</tbody>
</table>

(4 rows)
Connector Framework Service

The Connector Framework Service (CFS) allows secure indexing of documents from IDOL to the Vertica Analytic Database. Access Control Lists determine which users have permissions to access documents. Documents transferred from IDOL are stored in a Flex table (Using Flex Tables).

The dbadmin creates a view from this flex table (see Views). Users access the IDOL data from these views.

CFS Components

Use the following CFS components to implement the service on the Vertica:

- IDOL document metadata
- Security key and authorization functions
- SQL statement and Security Parameter
- CFS Configuration file

See Implementing CFS

Document Metadata

Vertica Analytic Database stores IDOL document metadata in a flex table. Set the name of the flex table with the TableName parameter in the CFS configuration file (see Modify the CFS Configuration File). The metadata includes the following:

- AUTONOMYMETADATA (Mandatory)—An alphanumeric designation for the ACL designated for the document.
- DREFIELD—Assigns permission levels to users and groups for accessing IDOL documents.
- DRETITLE—The document title.

For information see.
Security Key and Authorization Functions

As the dbadmin user you can assign each user a unique Security Information String (SIS). The strings are encrypted with a key stored in /idol/community/key. This key uses the following functions:

- INSTALL_COMMUNITY_KEY
- DESCRIBE_COMMUNITY_KEY
- DELETE_COMMUNITY_KEY

In addition to the preceding functions, the authorization function IDOL_CHECK_ACL verifies that the user has access to data in the view.

These functions reside in the v_idol schema contained in the idollib library installed with the CFS. When you run the functions, you must use v_idol in the command, for example:

```
=> SELECT v_idol.DELETE_COMMUNITY_KEY();
```

SQL Statement and Security Parameter

As the dbadmin user set the Security Information String for a specific user using:

- ALTER SESSION

- IdolSecurityInfo user-defined session parameter.

CFS Configuration File

You must index IDOL metadata in Vertica Analytic Database to be available for queries. See Implementing CFS
Implementing CFS

After Vertica Analytic Database ingests documents from IDOL, you can implement CFS to secure those documents. Implementing the security requires action from both the Vertica database administrator and the user who runs queries.

The database administrator must:

- **Modify the CFS Configuration File**
- **Create a View** from the flex table where the ingested IDOL data resides
- **Install the Security Key** used to decrypt the Security Information String
- **Verify CFS implementation success**

Upon CFS implementation, the user can **Query the IDOL Data**:

Modify the CFS Configuration File

Set the following in the CFS configuration file to have CFS automatically index the metadata:

1. In the `[Indexing]` section, set the `IndexerSections` parameter to `vertica`:

   ```
   [Indexing]
   IndexerSections=vertica
   IndexBatchSize=1
   IndexTimeInterval=30
   ```

2. Create a new section with the same name you entered in the `IndexerSections` parameter and enter the following parameters and keywords:

   ```
   [vertica]
   IndexerType=Library
   ConnectionString=Driver=Vertica;Server=123.456.478.900;Database=myDb;UID=dbadmin;PWD=password
   TableName=myFlexTable
   LibraryDirectory=./shared_library_indexers
   LibraryName=VerticaIndexer
   ```

The `VerticaIndexer` (LibraryName above) is part of CFS. To use this tool, you must install and configure the Vertica ODBC drivers on the same machine as CFS. CFS sends JSON-formatted data to the Flex table using ODBC. For more information, see Installing ODBC Drivers on Linux, Solaris, AIX, and HP-UX.
Create a View

The Vertica database administrator must create a view from the flex table where the ingested IDOL data resides. Users querying IDOL data do so using this view, thus preventing unauthorized access to the flex table.

This example creates a view from the idol_table:

```sql
=> CREATE VIEW idol_view as select * from idol_table where v_idol.idol_check_acl(acl, security_section, security_type);
```

You must then run GRANT privileges as follows:

1. Grant usage on the v_idol schema:

```sql
=> GRANT USAGE on schema v_idol to user1;
GRANT PRIVILEGE
```

2. Grant SELECT privileges on the view to any user who needs access:

```sql
=> GRANT SELECT on idol_view to user1;
GRANT PRIVILEGE
```

3. Grant usage on the library. This allows you to set UDSession parameters later:

```sql
=> GRANT USAGE on library v_idol.IdolLib to user1;
GRANT PRIVILEGE
```

4. Grant execute privileges on the acl-checking function.

```sql
=> GRANT EXECUTE on function v_idol.idol_check_acl(long varchar, long varchar, long varchar) to user1;
GRANT PRIVILEGE
```

Important: When granting EXECUTE privileges on a CFS function, ensure that you only grant the privilege to v_idol.idol_check_acl(). Never grant EXECUTE privileges to the following CFS functions:

- v_idol.install_community_key()
- v_idol.describe_community_key()
- v_idol.delete_community_key()
Install the Security Key

Before a user can access the view containing IDOL data, the database administrator must retrieve and install the security key. Store the security key in Vertica's Distributed File System (DFS) with the following command:

```
=> SELECT v_idol.install_community_key(USING PARAMETERS file_path='/home/user1/Downloads/IDOL/cli/aes.txt');
```

The Security Key is the same one used in IDOL. You can only install one key at a time. Installing another key overwrites the existing one.

The SecurityInfoKeys parameter in the following IDOL configuration files indicates the key you need to install:

- community/Community.cfg
- content/Content.cfg

After installing the key, run the following command to ensure it was correctly installed:

```
=> SELECT v_idol.describe_community_key();
```

```
daf3cc7d3c9368c2e78ed336aec6d8e75ee038c47aajc76f86a3bb5b197639ed43725a
(1 row)
```

See DELETE_COMMUNITY_KEY for information on deleting security keys.

Verify CFS implementation success

As a database administrator, do the following to verify the successful implementation of CFS:

1. Retrieve the Security Information String (SIS) from your organization's application. This is accomplished outside of Vertica.

2. Set the retrieved SIS for your current session:

```
=> ALTER SESSION set UDPARAMETER FOR v_idol.IdolLib IdolSecurityInfo = 'MzA0U08/+T59PHj6Pn64v/m8A';
```

3. Run the following ACL check function to see if it returns true only for rows to which you should have access:
Alternatively you can run the ACL check function and explicitly pass the SIS:

```sql
=> SELECT v_idol.idol_check_acl(acl, security_section, security_type) from cfs_table;
    idol_check_acl
  ----------------
     f
     t
   (2 rows)
```

**Query the IDOL Data**

A user querying IDOL must do the following:

1. **Retrieve the Security Information String (SIS) from your organization’s application.** This is accomplished outside of Vertica.

2. **Set the retrieved SIS for your current user session:**

   ```sql
   => ALTER SESSION set UDTPARAMETER FOR v_idol.IdolLib IdolSecurityInfo = 'MzA0U08/+T59PHj6Pn64v/m8A';
   ``

This example checks the v_idol.t flex table for the idol access control list, security section, and security type. All these parameters must match the metadata in the flex table.

3. **Run the query:**

   ```sql
   => SELECT * FROM idol_view;
   ```
Federal Information Processing Standard

When running on a certified FIPS-140-2 Red Hat 6.6 system, Vertica uses a certified OpenSSL FIPS 140-2 cryptographic module. This meets the security standards set by the National Institute of Standards and Technology (NIST) for Federal Agencies in the United States or other countries.

The standard specifies the security requirements that a cryptographic module needs in a system protecting sensitive information. For details on the standard see the Computer Security Resource Center.

**Note:** Vertica itself is not FIPS compliant but it is compatible with running on a FIPS-enabled system using FIPS resources.

For a list of FIPS prerequisites, see FIPS 140-2 Supported Platforms.

OpenSSL Behavior

Dynamic OpenSSL linking is a requirement for a FIPS implementation linking on the client and server. The Vertica server uses the OpenSSL that resides on the host system (version 1.0.1e as indicated in FIPS 140-2 Supported Platforms). OpenSSL dynamically links with LDAP and Kerberos.

For more information see Locate OpenSSL Libraries.

Libraries on CentOS 6.6 FIPS Systems

On a CentOS 6.6 FIPS system, Vertica runs only with the OpenSSL libraries `libcrypto.so.1.0.1e` and `libssl.so.1.0.1e`. Other versions of these libraries do not run on a FIPS system. This incompatibility occurs because the FIPS security policy checksums the library to which an application is linked and verifies that the library the application executes with the same checksum.
Library Versioning on Non-FIPS Systems

Be aware that on some non-FIPS systems, versioning anomalies can occur when you install a new version of OpenSSL. Sometimes, the default OpenSSL build procedure produces libraries with versions named 1.0.0. For Vertica to recognize that a library has a higher version number, you must provide the library name with a higher version number. For example, when installing OpenSSL version 1.0.1t, name the libraries libcrypto.so.1.0.1t or libssl.1.0.1t (symbolic links with these names are sufficient).

Install FIPS-enabled Vertica

The Vertica Analytic Database installation process determines if your system environment is FIPS compliant by checking the file /proc/sys/crypto/fips_enabled as follows:

```
$ sysctl crypto.fips_enabled
crypto.fips_enabled = 1
```

- If fips_enabled contains a 1, the host is FIPS enabled.
- If fips_enabled contains a 0, the host is not FIPS enabled.

If the host is FIPS enabled, the installation does the following:

- Verifies that OpenSSL resides in the appropriate area. If application does not exist before installation, the installer uses the OpenSSL provided by Vertica as the default. (OpenSSL is stored in /opt/vertica/lib).
- Runs a test to verify that Vertica was successfully configured for FIPS. If this test fails on any node, the installer fails.

For more information see Installing Vertica.

FIPS-Enabled Databases

Manually creating a new database on a FIPS-enabled Vertica Analytic Database requires a different approach than for a non-FIPS machine. Be aware of the following limitations:
You cannot create a FIPS-enabled database on a non-FIPS machine.

You cannot create a non-FIPS database on a FIPS-enabled machine.

Copying data generated with the MD5 hashing algorithm from a non-FIPS machine to a FIPS-enabled machine results in data corruption.

Implementing FIPS 140-2

Implementing FIPS 140-2 on your Vertica Analytic Database requires configuration on the server and client. The Vertica server uses FIPS-approved algorithms; however Vertica clients may be running on non-FIPS-approved systems. Therefore, you must implement FIPS 140-2 compliance from end to end.

For more information on implementing FIPS, see:

- Implement FIPS on the Server
- Implement FIPS on the Client

Implement FIPS on the Server

To implement FIPS on the Vertica server, you must:

- Generate a secure SSL certificate to establish a secure connection to the client.
- If necessary, set the LD_LIBRARY_PATH environment variable to locate the OpenSSL libraries.

RequireFIPS Parameter

Upon startup Vertica sets the RequireFIPS parameter on the server to reflect the FIPS state of the system, as follows:
Specify that FIPS is disabled:

```
RequireFIPS = 0
```

Specify that FIPS is enabled:

```
RequireFIPS = 1
```

The value of RequireFIPS matches the value of crypto.fips_enabled file. See Install FIPS-enabled Vertica.

Verify the value of the RequireFIPS parameter as follows:

```
=> SELECT get_config_parameter('RequireFIPS');
get_config_parameter
----------------------
0
(1 row)
```

Depending on the FIPS state, the following behaviors can occur:

- If the file `/proc/sys/crypto/fips_enabled` exists and contains a 1 (FIPS-enabled), Vertica sets RequireFIPS to 1. Modifying RequireFIPS with `ALTER DATABASE (RequireFIPS = 0)` generates an error.

- If the file `/proc/sys/crypto/fips_enabled` does not exist, or exists and contains a 0 (non-FIPS), Vertica automatically sets RequireFIPS to 0. Modifying RequireFIPS with `ALTER DATABASE (RequireFIPS = 1)` generates an error.

- If the FIPS state of a node, as determined from the existence of `/proc/sys/crypto/fips_enabled`, differs from the state received from the cluster initiator, the node fails. This behavior prevents the creation of clusters of mixed FIPS and non-FIPS systems.

**Important:** If you attempt to restore a FIPS-enabled node to a non-FIPS cluster, the restore process fails.

**Locate OpenSSL Libraries**

Vertica must find and load the correct OpenSSL libraries, `libcrypto.so.1.0.1.e` and `libssl.so.1.0.1.e`. To do so, it searches the system directory where the libraries reside. If the SSL libraries are not found, Vertica uses its own openssl libraries that reside under `/opt/vertica/lib`. 
Note: If you do not use admintools to start Vertica, or have conflicting libraries in your system, you must manually set LD_LIBRARY_PATH with `/opt/vertica/lib` appearing first in the list. When admintools starts or reboots Vertica, the path is set automatically.

Secure Client-Server Connection

Vertica uses TLS 1.2 to support the server-client connection for a FIPS-enabled system. This specification includes using a server certificate issued by a Certificate Authority.

Note: Using TLS 1.2 prevents you from using the MD5 algorithm for hashing passwords. Vertica accepts only AuthenticatedClearTextPasswords hashed by SHA-512. Users with MD5 passwords must migrate to SHA-512 passwords. For more information, see Upgrade Considerations for Hash Authentication.

For instructions on generating a self-signed certificate see Generating SSL Certificates and Keys. After generating a certificate, you need to distribute it to all hosts on the cluster. See Distributing Certificates and Keys. This distribution stores the certificate in the SSLCertificate parameter and the private key in the SSLPrivateKey parameter. For more information see Security Parameters.

Implement FIPS on the Client

Vertica provides a FIPS-compliant client driver, which you can install on a FIPS-enabled system. The 64-bit client includes vsq1 and ODBC drivers.

For information on the FIPS client, and installation, refer to the following

- Installing the FIPS Client Driver for ODBC and vsq1.
- Installing the FIPS Client Driver for JDBC

FIPS 140-2 Compliance Statement

Contents

1. Summary
2. Overview
1. Summary

Vertica complies with Federal Information Processing Standard 140-2 (FIPS 140-2), which defines the technical requirements to be used by Federal Agencies when these organizations specify cryptographic-based security systems for protection of sensitive or valuable data. The compliance of Vertica with FIPS 140-2 is ensured by: 1) Integrating validated and NIST-certified third party cryptographic module(s), and using the module(s) as the only provider(s) of cryptographic services; 2) Using FIPS-approved cryptographic functions; 3) Using FIPS-approved and NIST-validated technologies applicable for Vertica design, implementation and operation.

2. Overview

a. About Vertica

- Vertica is a high performance relational database management system used for advanced analytics applications. Its performance and scale is achieved through a columnar storage and execution architecture that offers a massively parallel processing solution. Aggressive encoding and compression allows Vertica analytics to perform by reducing CPU, memory and disk I/O Processing times.

- For more details about Vertica and its usage, see Vertica Concepts.

b. About FIPS 140-2

FIPS (Federal Information Processing Standard) 140-2, *Security requirements for cryptographic modules*, is the Federal standard for proper cryptography for computer systems purchased by the government.


The benefits of using FIPS 140-2 validated crypto module is that the crypto algorithms are deemed appropriate and that they perform the encrypt/decrypt/hash functions correctly. The standard specifies the security requirements for cryptographic modules utilized within a
security system that protects sensitive or valuable data. The requirements can be found in the following documents:

- SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES
- Annex A: Approved Security Functions for FIPS PUB 140-2, Security Requirements for Cryptographic Modules

3. Vertica and FIPS 140-2

FIPS 140-2 validated third party module

Vertica conforms with FIPS 140-2 Level 1 compliance by dynamically linking to the FIPS 140-2 approved OpenSSL cryptographic module provided by the Operating System, which in our initial release is RedHat Enterprise Linux 6.6 OpenSSL Module.

Vertica can be configured to operate in FIPS-compliant mode ensuring its functions and procedures like SSL/TLS connections, which require cryptography (secure hash, encryption, digital signatures, etc.) makes use of the crypto services provided by RedHat Enterprise 6.6 OpenSSL module Version 3.0 which is validated for FIPS 140-2. If you are not running on RedHat 6.6 you will not be able to run Vertica on FIPS mode. The assurance that Vertica is using the right FIPS 140-2 encryption modules is managed at the operating system level by RedHat’s implementation.

Vertica checks the OS level flag setting /proc/sys/crypto/fips_enabled to kick off Vertica’s FIPS mode installation. Further details about how to install and configure Vertica and its components to conform to FIPS 140-2 standard appear in the installation and security guides:

- Installing Vertica with the Installation Script
- Federal Information Processing Standard

Modes of Operation

Vertica Server operates in one of two modes determined by the OS configuration.

- FIPS-compliant mode – supports FIPS 140-2 compliant cryptographic functions. In this mode, all cryptographic functions, default algorithms and key lengths are bound to those allowed by FIPS 140-2.
- Standard mode – non-FIPS 140-2 compliant mode which utilizes all existing Vertica cryptography functions.
**TLS/SSL3.x**

All the Vertica client/server communications can be secured with FIPS-compliant Transport Layer Security TLS1.2/SSL3.1 or higher. It is relying on FIPS 140-2 approved hash algorithms and ciphers.

- TLS handshake, key negotiation and authentication provides data integrity and uses secure hash and FIPS 140-2 approved cryptography and digital signature.
- TLS encryption of data in transit provides confidentiality and making use of FIPS 140-2 approved cryptography.

**Secure Hash**

Per FIPS 140-2 standards, Vertica, in the FIPS 140-2 compliant mode, can be configured to use only the SHA-512 algorithm.

**FIPS 140-2 Architecture**

Vertica is a relational database system that is comprised of a client component and a server component. On the Client Side, we offer a suite of drivers for host clients to access the Vertica Server Side component. Both client and server Vertica components conform to FIPS 140-2 Level 1 compliance by dynamically linking to the FIPS 140-2 approved OpenSSL cryptographic module provided by RedHat Enterprise Linux 6.6 OpenSSL Module.

---

**Supported platforms**

Supported operating systems (both client and server):

- RedHat Enterprise Linux 6.6 64 bit

Supported client drivers:

- ODBC only
**Design Assurance**

Vertica uses the security provider Red Hat Enterprise Linux 6.6 OpenSSL Module v3.0. This is the only supported security provider for FIPS 140-2.

Once you have configured Vertica to be compliant with FIPS 140-2, you cannot revert back to the standard configuration unless you disable FIPS 140-2 at the operating system level. Please reference the following documentation section for considerations:

- Implement FIPS on the Server
- Implement FIPS on the Client
System Table Restriction and Access

Two functions let you restrict and open access to system tables for a given session:

- **RESTRICT_SYSTEM_TABLES_ACCESS** restricts access to non-superuser-only tables that are not accessible during lockdown.

- **RELEASE_SYSTEM_TABLES_ACCESS** allows access to non-superuser-only tables that are not accessible during lockdown.
Extending Vertica

You can extend Vertica to perform new operations or handle new types of data. There are several types of extensions:

- External procedures let you execute external scripts or programs that are installed on a host in your database cluster.

- User-Defined SQL Functions let you store frequently-used SQL expressions. They can help you simplify and standardize your SQL scripts.

- User-Defined Extensions (UDxs) let you develop your own analytic or data-loading tools using the C++, Python, Java, and R programming languages. They are useful when the type of data processing you want to perform is difficult or slow using SQL.

The following sections explain how to use extensions that have already been written:

- Using External Procedures
- Using User-Defined SQL Functions
- Using User-Defined Extensions

Developing User-Defined Extensions (UDxs) explains how to write new UDxs, including the following types:
- **Aggregate Functions (UDAFs)**
- **Analytic Functions (UDAnFs)**
- **Scalar Functions (UDSFs)**
- **Transform Functions (UDTFs)**
- **Load (UDLs)**
Using External Procedures

The external procedure feature lets you call a script or executable program on a host in your database cluster from within Vertica. You can pass literal values to this external procedure as arguments. The external procedure cannot communicate back to Vertica.

This chapter explains how to create, install, and use external procedures.
Implementing External Procedures

To implement an external procedure:

   
   See Requirements for External Procedures.

2. Enable the set-user-ID (SUID), user execute, and group execute attributes for the file. The file must either be readable by the dbadmin or the file owner's password must be given with the Administration Tools install_procedure command.

3. Install the external procedure executable file.

4. Create the external procedure in Vertica.

Once a procedure is created in Vertica, you can execute or drop it, but you cannot alter it.

Requirements for External Procedures

External procedures have requirements regarding their attributes, where you store them, and how you handle their output. You should also be cognizant of their resource usage.

Procedure File Attributes

The procedure file cannot be owned by root. It must have the set-user-ID (SUID), user execute, and group execute attributes set. If it is not readable by the Linux database administrator user, then the owner's password will have to be specified when installing the procedure.

Handling Procedure Output

Vertica does not provide a facility for handling procedure output. Therefore, you must make your own arrangements for handling procedure output, which should include writing error, logging, and program information directly to files that you manage.
Handling Resource Usage

The Vertica resource manager is unaware of resources used by external procedures. Additionally, Vertica is intended to be the only major process running on your system. If your external procedure is resource intensive, it could affect the performance and stability of Vertica. Consider the types of external procedures you create and when you run them. For example, you might run a resource-intensive procedure during off hours.

Sample Procedure File

```bash
#!/bin/bash
echo "hello planet argument: $1" >> /tmp/myprocedure.log
```

Installing External Procedure Executable Files

To install an external procedure, use the Administration Tools through either menu or the command line.

Using the Admin Tools Menus

a. Run the Administration Tools.

```
$ /opt/vertica/bin/adminTools
```

b. On the AdminTools Main Menu, click Configuration Menu, and then click OK.

c. On the Configuration Menu, click Install External Procedure and then click OK.

d. Select the database on which you want to install the external procedure.

e. Either select the file to install or manually type the complete file path, and then click OK.

f. If you are not the superuser, you are prompted to enter your password and click OK.

The Administration Tools automatically create the `<database_catalog_path>/procedures` directory on each node in the database and installs the external
procedure in these directories for you.

g. Click OK in the dialog that indicates that the installation was successful.

Command Line

If you use the command line, be sure to specify the full path to the procedure file and the password of the Linux user who owns the procedure file;

For example:

```bash
$ admintools -t install_procedure -d vmartdb -f /scratch/helloworld.sh -p ownerpassword
Installating external procedure...
External procedure installed
```

Once you have installed an external procedure, you need to make Vertica aware of it. To do so, use the `CREATE PROCEDURE` statement, but review Creating External Procedures first.

Creating External Procedures

Once you have installed an external procedure, you need to make Vertica aware of it. To do so, use the `CREATE PROCEDURE` statement.

Only a superuser can create an external procedure. Initially, only superusers can execute an external procedure. However, a superuser can grant the right to execute a stored procedure to a user on the operating system. (See `GRANT (Procedure)`.)

Once created, a procedure is listed in the `V_CATALOG.USER_PROCEDURES` system table. Users can see only those procedures that they have been granted the privilege to execute.

Example

This example creates a procedure named `helloplanet` for the `helloplanet.sh` external procedure file. This file accepts one `VARCHAR` argument. The sample code is provided in Requirements for External Procedures.

```sql
=> CREATE PROCEDURE helloplanet(arg1 VARCHAR) AS 'helloplanet.sh' LANGUAGE 'external'
   USER 'dbadmin';
```

This example creates a procedure named `proctest` for the `copy_vertica_database.sh` script. This script copies a database from one cluster to another, and it is included in the server RPM located in the `/opt/vertica/scripts` directory.
CREATE PROCEDURE proctest(shosts VARCHAR, thosts VARCHAR, dbdir VARCHAR)
AS 'copy_vertica_database.sh' LANGUAGE 'external' USER 'dbadmin';

Overloading External Procedures
You can create multiple external procedures with the same name as long as they have a different signature (accept a different set of arguments). For example, you can overload the helloplanet external procedure to also accept an integer value:

CREATE PROCEDURE helloplanet(arg1 INT) AS 'helloplanet.sh' LANGUAGE 'external'
USER 'dbadmin';

After executing the above statement, your database catalog will have two external procedures named helloplanet—one that accepts a VARCHAR argument and one that accepts an integer. When you call the external procedure, Vertica determines which procedure to call based on the arguments you passed in the procedure call.

See Also
- CREATE PROCEDURE
- GRANT (Procedure)

Executing External Procedures
Once you define a procedure through the CREATE PROCEDURE statement, you can use it as a meta command through a simple SELECT statement. Vertica does not support using procedures in more complex statements or in expressions.

The following example runs a procedure named helloplanet:

SELECT helloplanet('earthlings');
helloplanet
---------
0
(1 row)

The following example runs a procedure named proctest. This procedure references the copy_vertica_database.sh script that copies a database from one cluster to another. It is installed by the server RPM in the /opt/vertica/scripts directory.
=> SELECT proctest(
   '-s qa01',
   '-t rbench1',
   '-D /scratch_b/qa/PROC_TEST');

Note: External procedures have no direct access to database data. If available, use ODBC or JDBC for this purpose.

Procedures are executed on the initiating node. Vertica runs the procedure by forking and executing the program. Each procedure argument is passed to the executable file as a string. The parent fork process waits until the child process ends.

To stop execution, cancel the process by sending a cancel command (for example, CTRL+C) through the client. If the procedure program exits with an error, an error message with the exit status is returned.

Permissions

To execute an external procedure, the user needs:

- EXECUTE privilege on procedure
- USAGE privilege on schema that contains the procedure

See Also

- CREATE PROCEDURE
- External Procedure Privileges

Dropping External Procedures

Only a superuser can drop an external procedure. To drop the definition for an external procedure from Vertica, use the DROP PROCEDURE statement. Only the reference to the procedure is removed. The external file remains in the <database>/procedures directory on each node in the database.

Note: The definition Vertica uses for a procedure cannot be altered; it can only be dropped.
Example

=> DROP PROCEDURE helloplanet(arg1 varchar);

See Also

- DROP PROCEDURE
Using User-Defined SQL Functions

User-Defined SQL Functions let you define and store commonly-used SQL expressions as a function. User-Defined SQL Functions are useful for executing complex queries and combining Vertica built-in functions. You simply call the function name you assigned in your query.

A User-Defined SQL Function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

For syntax and parameters for the commands and system table discussed in this section, see the following topics in the SQL Reference Manual:

- CREATE FUNCTION
- ALTER FUNCTION (UDF)
- DROP FUNCTION
- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
- V_CATALOG.USER_FUNCTIONS

Creating User-Defined SQL Functions

A user-defined SQL function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.

To create a SQL function, the user must have CREATE privileges on the schema. To use a SQL function, the user must have USAGE privileges on the schema and EXECUTE privileges on the defined function.

This following statement creates a SQL function called myzeroifnull that accepts an INTEGER argument and returns an INTEGER result.

```sql
=> CREATE FUNCTION myzeroifnull(x INT) RETURN INT
   AS BEGIN
       RETURN (CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END);
   END;
```
You can use the new SQL function (myzeroifnull) anywhere you use an ordinary SQL expression. For example, create a simple table:

```sql
=> CREATE TABLE tabwnulls(col1 INT);
=> INSERT INTO tabwnulls VALUES(1);
=> INSERT INTO tabwnulls VALUES(NULL);
=> INSERT INTO tabwnulls VALUES(0);
=> SELECT * FROM tabwnulls;
```

```sql
<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
(3 rows)
```

Use the myzeroifnull function in a SELECT statement, where the function calls col1 from table tabwnulls:

```sql
=> SELECT myzeroifnull(col1) FROM tabwnulls;
```

```sql
myzeroifnull
-----------
  1
  0
  0
(3 rows)
```

Use the myzeroifnull function in the GROUP BY clause:

```sql
=> SELECT COUNT(*) FROM tabwnulls GROUP BY myzeroifnull(col1);
```

```sql
count
-----
  2
  1
(2 rows)
```

If you want to change a user-defined SQL function's body, use the CREATE OR REPLACE syntax. The following command modifies the CASE expression:

```sql
=> CREATE OR REPLACE FUNCTION myzeroifnull(x INT) RETURN INT
   AS BEGIN
     RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
   END;
```

To see how this information is stored in the Vertica catalog, see Viewing Information About SQL Functions in this guide.

See Also

- CREATE FUNCTION (SQL Functions)
- USER_FUNCTIONS
## Altering and Dropping User-Defined SQL Functions

Vertica allows multiple functions to share the same name with different argument types. Therefore, if you try to alter or drop a SQL function without specifying the argument data type, the system returns an error message to prevent you from dropping the wrong function:

```sql
=> DROP FUNCTION myzeroifnull();
ROLLBACK: Function with specified name and parameters does not exist: myzeroifnull
```

Note: Only a superuser or owner can alter or drop a SQL Function.

### Altering a User-Defined SQL Function

The `ALTER FUNCTION (UDF)` command lets you assign a new name to a user-defined function, as well as move it to a different schema.

In the previous topic, you created a SQL function called `myzeroifnull`. The following command renames the `myzeroifnull` function to `zerowhennull`:

```sql
=> ALTER FUNCTION myzeroifnull(x INT) RENAME TO zerowhennull;
ALTER FUNCTION
```

This next command moves the renamed function into a new schema called `macros`:

```sql
=> ALTER FUNCTION zerowhennull(x INT) SET SCHEMA macros;
ALTER FUNCTION
```

### Dropping a SQL Function

The `DROP FUNCTION` command drops a SQL function from the Vertica catalog.

Like with `ALTER FUNCTION`, you must specify the argument data type or the system returns the following error message:

```sql
=> DROP FUNCTION zerowhennull();
ROLLBACK: Function with specified name and parameters does not exist: zerowhennull
```

Specify the argument type:
DROP FUNCTION macros.zerowhennull(x INT);

Vertica does not check for dependencies, so if you drop a SQL function where other objects reference it (such as views or other SQL Functions), Vertica returns an error when those objects are used, not when the function is dropped.

Tip: To view a list of all user-defined SQL functions on which you have EXECUTE privileges, (which also returns their argument types), query the V_CATALOG.USER_FUNCTIONS system table.

See Also

- ALTER FUNCTION (UDF)
- DROP FUNCTION

Managing Access to SQL Functions

Before a user can execute a user-defined SQL function, he or she must have USAGE privileges on the schema and EXECUTE privileges on the defined function. Only the superuser or owner can grant/revoke EXECUTE usage on a function.

To grant EXECUTE privileges to user Fred on the myzeroifnull function:

```sql
=> GRANT EXECUTE ON FUNCTION myzeroifnull (x INT) TO Fred;
```

To revoke EXECUTE privileges from user Fred on the myzeroifnull function:

```sql
=> REVOKE EXECUTE ON FUNCTION myzeroifnull (x INT) FROM Fred;
```

See Also

- GRANT (User Defined Extension)
- REVOKE (User Defined Extension)
Viewing Information About User-Defined SQL Functions

You can access information about any User-Defined SQL Functions on which you have EXECUTE privileges. This information is available in the system table `V_CATALOG.USER_FUNCTIONS` and from the vsql meta-command `\df`.

To view all of the User-Defined SQL Functions on which you have EXECUTE privileges, query the `USER_FUNCTIONS` table:

```sql
=> SELECT * FROM USER_FUNCTIONS;
```

<table>
<thead>
<tr>
<th>schema_name</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_name</td>
<td>myzeroifnull</td>
</tr>
<tr>
<td>function_return_type</td>
<td>Integer</td>
</tr>
<tr>
<td>function_argument_type</td>
<td>x Integer</td>
</tr>
<tr>
<td>function_definition</td>
<td>RETURN CASE WHEN (x IS NOT NULL) THEN x ELSE 0 END</td>
</tr>
<tr>
<td>volatility</td>
<td>immutable</td>
</tr>
<tr>
<td>is_strict</td>
<td>f</td>
</tr>
</tbody>
</table>

If you want to change a User-Defined SQL Function's body, use the CREATE OR REPLACE syntax. The following command modifies the CASE expression:

```sql
=> CREATE OR REPLACE FUNCTION myzeroifnull(x INT) RETURN INT
  AS BEGIN
    RETURN (CASE WHEN (x IS NULL) THEN 0 ELSE x END);
  END;
```

Now when you query the `USER_FUNCTIONS` table, you can see the changes in the `function_definition` column:

```sql
=> SELECT * FROM USER_FUNCTIONS;
```

<table>
<thead>
<tr>
<th>schema_name</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>function_name</td>
<td>myzeroifnull</td>
</tr>
<tr>
<td>function_return_type</td>
<td>Integer</td>
</tr>
<tr>
<td>function_argument_type</td>
<td>x Integer</td>
</tr>
<tr>
<td>function_definition</td>
<td>RETURN CASE WHEN (x IS NULL) THEN 0 ELSE x END</td>
</tr>
<tr>
<td>volatility</td>
<td>immutable</td>
</tr>
<tr>
<td>is_strict</td>
<td>f</td>
</tr>
</tbody>
</table>

If you use CREATE OR REPLACE syntax to change only the argument name or argument type (or both), the system maintains both versions of the function. For example, the following command tells the function to accept and return a numeric data type instead of an integer for the `myzeroifnull` function:
CREATE OR REPLACE FUNCTION myzeroifnull(z NUMERIC) RETURN NUMERIC
AS BEGIN
    RETURN (CASE WHEN (z IS NULL) THEN 0 ELSE z END);
END;

Now query the USER_FUNCTIONS table, and you can see the second instance of myzeroifnull in Record 2, as well as the changes in the function_return_type, function_argument_type, and function_definition columns.

Note: Record 1 still holds the original definition for the myzeroifnull function:

Because Vertica allows functions to share the same name with different argument types, you must specify the argument type when you alter or drop a function. If you do not, the system returns an error message:

=> DROP FUNCTION myzeroifnull();
ROLLBACK: Function with specified name and parameters does not exist: myzeroifnull

See Also

- USER_FUNCTIONS

Migrating Built-In SQL Functions

If you have built-in SQL functions from another RDBMS that do not map to a Vertica-supported function, you can migrate them into your Vertica database by using a user-defined SQL function.
The example scripts below show how to create user-defined functions for the following DB2 built-in functions:

- **UCASE()**
- **LCASE()**
- **LOCATE()**
- **POSSTR()**

**UCASE()**

This script creates a user-defined SQL function for the `UCASE()` function:

```sql
=> CREATE OR REPLACE FUNCTION UCASE (x VARCHAR)
   RETURN VARCHAR
   AS BEGIN
   RETURN UPPER(x);
   END;
```

**LCASE()**

This script creates a user-defined SQL function for the `LCASE()` function:

```sql
=> CREATE OR REPLACE FUNCTION LCASE (x VARCHAR)
   RETURN VARCHAR
   AS BEGIN
   RETURN LOWER(x);
   END;
```

**LOCATE()**

This script creates a user-defined SQL function for the `LOCATE()` function:

```sql
=> CREATE OR REPLACE FUNCTION LOCATE(a VARCHAR, b VARCHAR)
   RETURN INT
   AS BEGIN
   RETURN POSITION(a IN b);
   END;
```
POSSTR()

This script creates a user-defined SQL function for the POSSTR() function:

=> CREATE OR REPLACE FUNCTION POSSTR(a VARCHAR, b VARCHAR)
   RETURN INT
   AS BEGIN
   RETURN POSITION(b IN a);
   END;
Using User-Defined Extensions

A User-Defined Extension (abbreviated as UDx) is a component that adds new abilities to the Vertica Analytics Platform. UDxs provide features such as new types of data analysis and the ability to parse and load new types of data.

This section provides an overview of how to install and use a UDx. If you are using a UDx developed by a third party, you should consult any documentation that accompanies it for detailed installation and usage instructions.

Loading UDxs

User-Defined Extensions (UDxs) are contained in libraries. A library can contain multiple UDxs. To add UDxs to Vertica you must:

1. Load the library (once per library).
2. Declare each UDx (once per UDx).

If you are using UDxs written in Java, you must also set up a Java runtime environment. See Installing Java on Vertica Hosts.

Loading Libraries

To deploy a library to your Vertica database:

1. Copy the UDx shared library file (.so), Python file, or JAR that contains your function to a node on your Vertica cluster. You do not need to copy it to every node.
2. Connect to the node where you copied the library (for example, using vsql).
3. Add your library to the database catalog using the CREATE LIBRARY statement.

```
=> CREATE LIBRARY libname AS '/path_to_lib/filename'
     LANGUAGE 'language';
```

libname is the name you want to use to reference the library. path_to_lib/filename is the fully-qualified path to the library or JAR file you copied to the host. language is the implementation language.
For example, if you created a JAR file named TokenizeStringLib.jar and copied it to the dbadmin account's home directory, you would use this command to load the library:

```sql
=> CREATE LIBRARY tokenizelib AS '/home/dbadmin/TokenizeStringLib.jar'
    LANGUAGE 'Java';
```

You can load any number of libraries into Vertica.

## Declaring Individual UDx

Once the library is loaded, define individual UDx using SQL statements such as `CREATE FUNCTION` and `CREATE SOURCE`. These statements assign SQL function names to the extension classes in the library. They add the UDx to the database catalog and remain available after a database restart.

The statement you use depends on the type of UDx you are declaring, as shown in the following table.

<table>
<thead>
<tr>
<th>UDx Type</th>
<th>SQL Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate Function (UDAF)</td>
<td><code>CREATE AGGREGATE FUNCTION</code></td>
</tr>
<tr>
<td>Analytic Function (UDAnF)</td>
<td><code>CREATE ANALYTIC FUNCTION</code></td>
</tr>
<tr>
<td>Scalar Function (UDSF)</td>
<td><code>CREATE FUNCTION (UDF)</code></td>
</tr>
<tr>
<td>Transform Function (UDTF)</td>
<td><code>CREATE TRANSFORM FUNCTION</code></td>
</tr>
<tr>
<td>Load (UDL): Source</td>
<td><code>CREATE SOURCE</code></td>
</tr>
<tr>
<td>Load (UDL): Filter</td>
<td><code>CREATE FILTER</code></td>
</tr>
<tr>
<td>Load (UDL): Parser</td>
<td><code>CREATE PARSER</code></td>
</tr>
</tbody>
</table>

After you add the UDx to the database, you can use your extension within SQL statements. The database superuser can grant access privileges to the UDx for users. See `GRANT (User Defined Extension)` in the SQL Reference Manual for details.

When you call a UDx, Vertica creates an instance of the UDx class on each node in the cluster and provides it with the data it needs to process.
Installing Java on Vertica Hosts

If you are using UDxs written in Java, follow the instructions in this section.

You must install a Java Virtual Machine (JVM) on every host in your cluster in order for Vertica to be able to execute your Java UDxs.

Installing Java on your Vertica cluster is a two-step process:

1. Install a Java runtime on all of the hosts in your cluster.

2. Set the JavaBinaryForUDx configuration parameter to tell Vertica the location of the Java executable.

Installing a Java Runtime

For Java-based features, Vertica requires a 64-bit Java 6 (Java version 1.6) or later Java runtime. Vertica supports runtimes from either Oracle or OpenJDK. You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE.

Many Linux distributions include a package for the OpenJDK runtime. See your Linux distribution's documentation for information about installing and configuring OpenJDK.

To install the Oracle Java runtime, see the Java Standard Edition (SE) Download Page. You usually run the installation package as root in order to install it. See the download page for instructions.

Once you have installed a JVM on each host, ensure that the java command is in the search path and calls the correct JVM by running the command:

```
$ java -version
```

This command should print something similar to:

```
java version "1.8.0_102"
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

Note: Any previously installed Java VM on your hosts may interfere with a newly installed Java runtime. See your Linux distribution's documentation for instructions on configuring which JVM is the default. Unless absolutely required, you should uninstall any incompatible
version of Java before installing the Java 6 or Java 7 runtime.

Setting the JavaBinaryForUDx Configuration Parameter

The JavaBinaryForUDx configuration parameter tells Vertica where to look for the JRE to execute Java UDxs. After you have installed the JRE on all of the nodes in your cluster, set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example /usr/bin/java). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

$ which java
/usr/bin/java

If the java command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. Suppose you installed the JRE in /usr/java/default (which is where the installation package supplied by Oracle installs the Java 1.6 JRE). In this case the Java executable is /usr/java/default/bin/java.

You set the configuration parameter by executing the following statement as a database superuser:

```sql
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
```

See `ALTER DATABASE` for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the `CONFIGURATION_PARAMETERS` system table:

```sql
=> \x
Expanded display is on.
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'JavaBinaryForUDx';
-[ RECORD 1 ]-+----------------------------------+
<table>
<thead>
<tr>
<th>node_name</th>
<th>parameter_name</th>
<th>current_value</th>
<th>default_value</th>
<th>change_under_support_guidance</th>
<th>change_requires_restart</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBinaryForUDx</td>
<td>/usr/bin/java</td>
<td></td>
<td>f</td>
<td>f</td>
<td>Path to the java binary for executing UDx written in Java</td>
</tr>
</tbody>
</table>
```

Once you have set the configuration parameter, Vertica can find the Java executable on each node in your cluster.

Note: Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the Java executable is installed in the same...
path on all of the hosts in the cluster.

Notes on Individual UDx Types

Some UDx types have special considerations or restrictions.

Analytic Functions

UDAnFs do not support framing windows using ROWS.

As with Vertica's built-in analytic functions, UDAnFs cannot be used with Pattern Matching Functions.

Scalar Functions

If the result of applying a UDSF is an invalid record, COPY aborts the load even if CopyFaultTolerantExpressions is set to true.

Transform Functions

A query that includes a UDTF cannot contain:

- Any statements other than the SELECT statement containing the call to the UDTF and a PARTITION BY expression
- Any other analytic function
- A call to another UDTF
- A TIMESERIES clause
- A pattern matching clause
- A gap filling and interpolation clause
Load

Installing an untrusted UDL function can compromise the security of the server. UDx's can contain arbitrary code. In particular, UD Source functions can read data from any arbitrary location. It is up to the developer of the function to enforce proper security limitations. Superusers must not grant access to UDxs to untrusted users.

You cannot ALTER UDL functions.

Fenced Mode

User-Defined Extensions (UDxs) written in the C++ programming language have the option of running in unfenced mode, which means running directly within the Vertica process. Since they run within Vertica, unfenced UDxs have little overhead, and can perform almost as fast as Vertica's own built-in functions. However, since they run within Vertica directly, any bugs in their code (memory leaks, for example) can destabilize the main Vertica process that can bring one or more database nodes down.

You can instead opt to run most C++ UDxs in fenced mode, which runs the UDxs code outside of the main Vertica process in a separate zygote process. UDx code that crashes while running in fenced mode does not impact the core Vertica process. There is a small performance impact when running UDx code in fenced mode. On average, using fenced mode adds about 10% more time to execution compared to unfenced mode.

Fenced mode is currently available for all C++ UDxs with the exception of User-Defined Aggregates. All UDxs developed in the Python, R, and Java programming languages must run in fenced mode, since the Python, R, and Java runtimes cannot be directly run within the Vertica process.

Using fenced mode does not affect the development of your UDx. Fenced mode is enabled by default for UDx's that support fenced mode. The CREATE FUNCTION command can optionally be issued with the NOT FENCED modifier to disable fenced mode for the function. Also, you can enable or disable fenced mode on any fenced mode-supported C++ UDx by using the ALTER FUNCTION command.

About the Zygote Process

The Vertica zygote process starts when Vertica starts. Each node has a single zygote process. Side processes are created "on demand". The zygote listens for requests and spawns a UDx
side session that runs the UDx in fenced mode when a UDx is called by the user.

About Fenced Mode Logging:

UDx code that runs in fenced mode is logged in the UDxZygote.log and is stored in the UDxLogs directory in the catalog directory of Vertica. Log entries for the side process are denoted by the UDx language (for example, C++), node, and zygote process ID, and the UDxSideProcess ID.

For example, for the following processes...

```sql
-> SELECT * FROM UDx_FENCED_PROCESSES;
node_name | process_type | session_id | pid | port | status
----------|--------------|------------|-----|------|--------
v_vmart_node0001 | UDxZygoteProcess | | 27468 | 51900 | UP
v_vmart_node0001 | UDxSideProcess | localhost.localdoma-27465:0x800b | 5677 | 44123 | UP
```

... the corresponding log file displays:

```
2016-05-16 11:24:43.990 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 UDx side process started
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 Finished setting up signal handlers.
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 My port: 44123
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 My address: 0.0.0.0
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 Vertica port: 51900
11:24:43.996 [C++-localhost.localdoma-27465:0x800b-5677] 0x2b3ff17e7f00 Vertica address: 127.0.0.1
11:25:19.749 [C++-localhost.localdoma-27465:0x800b-5677] 0x41037940 Setting memory resource limit to -1
11:30:11.523 [C++-localhost.localdoma-27465:0x800b-5677] 0x41037940 Exiting UDx side process
```

The last line indicates that the side process was killed. In this case it was killed when the user session (vsql) closed.

About Fenced Mode Configuration Parameters

Fenced mode supports three configuration parameters:

- **FencedUDxMemoryLimitMB** - The maximum memory size, in MB, to use for Fenced Mode processes. The default is -1 (no limit). The side process is killed if this limit is exceeded.

- **ForceUDxFencedMode** - When set to 1, force all UDx's that support fenced mode to run in fenced mode even if their definition specified NOT FENCED. The default is 0 (disabled).

- **UDxFencedBlockTimeOut** - The maximum time, in seconds, that the Vertica server waits for a UDx to return before aborting with ERROR 3399. The default is 60.
See Also

- CREATE LIBRARY
- CREATE FUNCTION
- CREATE TRANSFORM FUNCTION
- CREATE ANALYTIC FUNCTION
- ALTER FUNCTION (UDF)
- UDX_FENCED PROCESSES
Updating UDx Libraries

There are two cases where you need to update libraries that you have already deployed:

- When you have upgraded Vertica to a new version that contains changes to the SDK API. For your libraries to work with the new server version, you need to recompile them with new version of the SDK. See UDx Library Compatibility with New Server Versions for more information.

- When you have made changes to your UDxs and you want to deploy these changes. Before updating your UDx library, you need to determine if you have changed the signature of any of the functions contained in the library. If you have, you need to drop the functions from the Vertica catalog before you update the library.

UDx Library Compatibility with New Server Versions

The Vertica SDK defines an application programming interface (API) that UDxs use to interact with the database. When a developer compiles his or her UDx code, it is linked to the SDK code to form a library. This library is only compatible with Vertica servers that support the version of the SDK API used to compile the code. The library and servers that share the same API version are compatible on a binary level (referred to as "binary compatible").

The Vertica server returns an error message if you attempt to load a library that is not binary compatible with it. Similarly, if you upgrade your Vertica server to a version that supports a new SDK API, any existing UDx that relies on newly-incompatible libraries returns an error messages when you call it:

```
ERROR 2858: Could not find function definition
HINT: This usually happens due to missing or corrupt libraries, libraries built with the wrong SDK version, or due to a concurrent session dropping the library or function. Try recreating the library and function
```

To resolve this issue, you must install UDx libraries that have been recompiled with correct version of the SDK.

New versions of the Vertica server do not always change the SDK API version. The SDK API version changes whenever OpenText changes the components that make up the SDK. If the SDK API does not change in a new version of the server, then the old libraries remain compatible with the new server.
The SDK API almost always changes between major releases of Vertica (for example between versions 6.0 and 6.1), as OpenText expands the SDK’s features in each major release. In addition, OpenText reserves the right to change the SDK API in the first service pack release (for example between (from version 9.0.0 to 9.0.1) to address any issues found in the initial release of a major version. After the first service pack, the SDK API is frozen and will not change for rest of the service packs for that major release (for example, 9.0.2, 9.0.3, and so forth).

These policies mean that:

- You need to update UDx libraries when you upgrade between major versions. For example, if you upgrade from version 7.0.1 to 7.1.0 you need to update your UDx libraries.

- You may need to update the UDx libraries if you upgrade from the first version of a major release to a later service pack release. For example, if you upgrade the server from version 7.1.0 to 7.1.2, you may need to update your UDx libraries. Version 7.1.1 (the first service pack release) may have SDK changes that make your libraries compiled with the 7.1.0 SDK incompatible with the Vertica 7.1.2 server.

- You do not need to update your UDx libraries when upgrading from one service pack to another. For example, if you upgrade from version 7.1.1 to 7.1.2, you do not need to update your UDx libraries. The SDK API version is the same between there versions and any later 7.1.x service pack releases.

Note: Since the R language is interpreted, a UDx written in R is not linked to the Vertica SDK. There is no binary compatibility that has to be maintenanted from one version to another. Therefore, changes to the Vertica SDK between Vertica versions do not make your R-based UDx libraries incompatible with a new server version. An R-based UDx only becomes incompatible if the APIs uses in the SDK actually change. For example, if the number of arguments to an API call changes, the UDx written in R has to be changed to use the new number of arguments.

Pre-Upgrade Steps

Before upgrading your Vertica server, consider whether you have any UDx libraries that may be incompatible with the new version. Consult the release notes of the new server version to determine whether the SDK API has changed between the version of Vertica server you currently have installed and the new version. As mentioned previously, only upgrades from a previous major version or from the initial release of a major version to a service pack release can cause your currently-loaded UDx libraries to become incompatible with the server.

Any UDx libraries that are incompatible with the new version of the Vertica server must be recompiled. If you got the UDx library from a third party (such as through the Vertica
Marketplace), you need to see if a new version has been released. If so, then download the UDx library and deploy it after you have upgraded the server (see Deploying A New Version of Your UDx Library).

If you developed the UDx yourself (or if whoever developed it supplied the UDx's source code) you must:

1. Recompile your UDx library using the new version of the Vertica SDK. See Compiling Your C++ Library or Compiling and Packaging a Java Library for more information.

2. Deploy the new version of your library. See Deploying A New Version of Your UDx Library.

Determining If a UDx Signature Has Changed

You need to be careful when making changes to UDx libraries that contain functions you have already deployed in your Vertica database. When you deploy a new version of your UDx library, Vertica does not ensure that the signatures of the functions that are defined in the library match the signature of the function that is already defined in the Vertica catalog. If you have changed the signature of a UDx in the library then update the library in the Vertica database, calls to the altered UDx will produce errors.

Making any of the following changes to a UDx alters its signature:

- Changing the number of arguments accepted or the data type of any argument accepted by your function (not including polymorphic functions).

- Changing the number or data types of any return values or output columns.

- Changing the name of the factory class that Vertica uses to create an instance of your function code.

- Changing the null handling or volatility behavior of your function.

- Removed the function's factory class from the library completely.

The following changes do not alter the signature of your function, and do not require you to drop the function before updating the library:

- Changing the number or type of arguments handled by a polymorphic function. Vertica does not process the arguments the user passes to a polymorphic function.

- Changing the name, data type, or number of parameters accepted by your function. The parameters your function accepts are not determined by the function signature. Instead,
Vertica passes all of the parameters the user included in the function call, and your function processes them at runtime. See UDx Parameters for more information about parameters.

- Changing any of the internal processing performed by your function.
- Adding new UDxs to the library.

After you drop any functions whose signatures have changed, you load the new library file, then re-create your altered functions. If you have not made any changes to the signature of your UDxs, you can just update the library file in your Vertica database without having to drop or alter your function definitions. As long as the UDx definitions in the Vertica catalog match the signatures of the functions in your library, function calls will work transparently after you have updated the library. See Deploying A New Version of Your UDx Library.

Deploying A New Version of Your UDx Library

You need to deploy a new version of your UDx library if:

- You have made changes to the library that you now want to roll out to your Vertica database.
- You have upgraded your Vertica to a new version whose SDK is incompatible with the previous version.

The process of deploying a new version of your library is similar to deploying it initially.

1. If you are deploying a UDx library developed in C++ or Java, you must compile it with the current version of the Vertica SDK.

2. Copy your UDx's library file (a .so file for libraries developed in C++, a .py file for libraries developed in Python, or a .jar file for libraries developed in Java) or R source file to a host in your Vertica database.

3. Connect to the host using vsql.

4. If you have changed the signature of any of the UDxs in the shared library, you must drop them using DROP statements such as DROP FUNCTION or DROP SOURCE. If you are unsure whether any of the signatures of your functions have changed, see Determining If a UDx Signature Has Changed.
Note: If all of the UDx signatures in your library have changed, you may find it more convenient to drop the library using the DROP LIBRARY statement with the CASCADE option to drop the library and all of the functions and loaders that reference it. Dropping the library can save you the time it would take to drop each UDx individually. You can then reload the library and recreate all of the extensions using the same process you used to deploy the library in the first place. See CREATE LIBRARY in the SQL Reference Manual.

5. Use the ALTER LIBRARY statement to update the UDx library definition with the file you copied in step 1. For example, if you want to update the library named ScalarFunctions with a file named ScalarFunctions-2.0.so in the dbadmin user’s home directory, you could use the command:

```sql
=> ALTER LIBRARY ScalarFunctions AS '/home/dbadmin/ScalarFunctions-2.0.so';
```

Once you have updated the UDx library definition to use the new version of your shared library, the UDxs that are defined using classes in your UDx library begin using the new shared library file without any further changes.

6. If you had to drop any functions in step 4, recreate them using the new signature defined by the factory classes in your library. See CREATE FUNCTION Statements in the SQL Reference Manual.
Listing the UDxs Contained in a Library

Once a library has been loaded using the **CREATE LIBRARY** statement, you can find the UDxs and UDLs it contains by querying the **USER_LIBRARY_MANIFEST** system table:

```sql
=> CREATE LIBRARY ScalarFunctions AS '/home/dbadmin/ScalarFunctions.so';
CREATE LIBRARY => \\x
Expanded display is on.
=> SELECT * FROM USER_LIBRARY_MANIFEST WHERE lib_name = 'ScalarFunctions';
- [ RECORD 1 ]----------------------
schema_name | public
lib_name     | ScalarFunctions
lib_oid      | 45035996273792402
obj_name     | RemoveSpaceFactory
obj_type     | Scalar Function
arg_types    | Varchar
return_type  | Varchar
- [ RECORD 2 ]----------------------
schema_name | public
lib_name     | ScalarFunctions
lib_oid      | 45035996273792402
obj_name     | Div2intsInfo
obj_type     | Scalar Function
arg_types    | Integer, Integer
return_type  | Integer
- [ RECORD 3 ]----------------------
schema_name | public
lib_name     | ScalarFunctions
lib_oid      | 45035996273792402
obj_name     | Add2intsInfo
obj_type     | Scalar Function
arg_types    | Integer, Integer
return_type  | Integer
```

The `obj_name` column lists the factory classes contained in the library. These are the names you use to define UDxs and UDLs in the database catalog using statements such as **CREATE FUNCTION** and **CREATE SOURCE**.

Using Wildcards In Your UDx

Vertica supports wildcard * characters in the place of column names in user-defined functions. You can use wildcards when:
- Your query contains a table in the FROM clause
- You are using a Vertica-supported development language
- Your UDx is running in fenced or unfenced mode

**Supported SQL Statements**

The following SQL statements can accept wildcards:

- DELETE
- INSERT
- SELECT
- UPDATE

**Unsupported Configurations**

The following situations do not support wildcards:

- You cannot pass a wildcard in the OVER clause of a query
- You cannot us a wildcard with a DROP statement
- You cannot use wildcards with any other arguments

**Examples**

These examples show wildcards and user-defined functions in a range of data manipulation operations.

**DELETE statements:**

```sql
=> DELETE FROM tablename WHERE udf(tablename.*) = 5;
```

**INSERT statements:**
### INSERT INTO table1 SELECT udf(*) FROM table2;

### SELECT statements:

- `=> SELECT udf(*) FROM tablename;`  
- `=> SELECT udf(tablename.*) FROM tablename;`  
- `=> SELECT udf(f.*) FROM table f;`  
- `=> SELECT udf(*) FROM table1,table2;`  
- `=> SELECT udf1( udf2(*) ) FROM table1,table2;`  
- `=> SELECT udf( db.schema.table.*) FROM tablename;`  
- `=> SELECT udf(sub.*) FROM (select col1, col2 FROM table) sub;`  
- `=> SELECT x FROM tablename WHERE udf(*) = y;`  
- `=> WITH sub as (SELECT * FROM tablename) select x, udf(*) FROM sub;`  
- `=> SELECT udf( * using parameters x=1) FROM tablename;`  
- `=> SELECT udf(table1.*, table2.col2) FROM table1,table2;`  

### UPDATE statements:

- `=> UPDATE tablename set col1 = 4 FROM tablename WHERE udf(*) = 3;`
Developing User-Defined Extensions (UDxs)

User-Defined Extensions (UDxs) are functions contained in external shared libraries that are developed in C++, Python, Java, or R using the Vertica SDK. The external libraries are defined in the Vertica catalog using the CREATE LIBRARY statement. They are best suited for analytic operations that are difficult to perform in SQL, or need to be performed frequently enough that their speed is a major concern.

The primary strengths of UDxs are:

- They can be used anywhere an internal function can be used.
- They take full advantage of Vertica's distributed computing features. The extensions usually execute in parallel on each node in the cluster.
- Vertica handles the distribution of the UDx library to the individual nodes. You only need to copy the library to the initiator node.
- All of the complicated aspects of developing a distributed piece of analytic code are handled for you by Vertica. Your main programming task is to read in data, process it, and then write it out using the Vertica SDK APIs.

There are a few things to keep in mind about developing UDxs:

- UDxs can be developed in the programming languages C++, Python, Java, and R. (Not all UDx types support all languages.)
- UDxs written in Java always run in Fenced Mode, because the Java Virtual Machine that executes Java programs cannot run directly within the Vertica process.
- UDxs written in Python and R always run in Fenced Mode
- UDxs developed in C++ have the option of running in unfenced mode, which means they load and run directly in the Vertica database process. This option provides the lowest overhead and highest speed. However, any bugs in the UDx's code can cause database instability. You must thoroughly test any UDxs you intend to run in unfenced mode before deploying them in a live environment. Consider whether the performance boost of running a C++ UDx unfenced is worth the potential database instability that a buggy UDx can cause.
• Because a UDx runs on the Vertica cluster, it can take processor time and memory away from the database processes. A UDx that consumes large amounts of computing resources can negatively impact database performance.

**Structure**

Each UDx type consists of two classes. The main class does the primary work (a transformation, an aggregation, and so on). The class usually has at least three methods: one to set up, one to tear down (release reserved resources), and one to do the actual work. Sometimes additional methods are defined.

The main processing method receives an instance of the ServerInterface class as an argument. This object is used by the underlying Vertica SDK code to make calls back into the Vertica process, for example to allocate memory. You can use this class to write to the server log during UDx execution.

The second class is a singleton factory. It defines one method that produces instances of the first class, and might define other methods to manage parameters.

When implementing a UDx you must subclass both classes.

**Conventions**

The C++, Python, and Java APIs are nearly identical. Where possible, this documentation describes these interfaces without respect to language. Documentation specific to C++, Python, or Java is covered in language-specific sections.

Because some documentation is language-independent, it is not always possible to use ideal, language-based terminology. This documentation uses the term "method" to refer to a Java method or a C++ member function.

**See Also**

Loading UDxs
Setting Up a Development Environment

Before you start developing your UDx, you need to configure your development environment. You can choose to develop your UDx on a node in a development Vertica database (not in a production environment) or on another machine.

To test, you need access to a (non-production) Vertica database. You can install a single-node Vertica database on your development machine for easier development.

You should develop your UDx code on the same Linux platform that you use on your Vertica database cluster. This will ensure that your UDx library is compatible with the Vertica version deployed on your cluster.

The following sections describe language-specific requirements.

C++ Requirements

At a minimum, you need to install the following on your development machine:

- `g++` and its associated tool chain such as ld. (Note: some Linux distributions package g++ separately from gcc.)

- A copy of the Vertica SDK. See Setting Up the C++ SDK for details.

*Note: The Vertica binaries are compiled using the default version of g++ installed on the supported Linux platforms. Vertica requires a minimum of gcc version 4.8.4. The default version of gcc on Ubuntu 16.04 is not compatible; see Requirements for Ubuntu 16.04.*

While not required, the following additional software packages can ease development:

- make, or some other build-management tool.

- gdb, or some other debugger.

- Valgrind, or similar tools that detect memory leaks.

- Although not currently required, future versions of Vertica may require that you compile with the flag `-std=c++11`.

If you want to use any third-party libraries (for example, statistical analysis libraries), you need to install them on your development machine. If you do not statically link these libraries into
your UDx library, you also have to install them on every node in the cluster. See Compiling Your C++ Library for details.

Requirements for Ubuntu 16.04

Ubuntu 16.04 ships with a compiler that is incompatible with the Vertica C++ SDK. To compile UDxs on this platform, you must install packages for 4.8 compatibility:

```bash
$ sudo apt-get remove g++
$ sudo apt-get remove gcc
$ sudo apt-get install gcc-4.8
$ sudo apt-get install g++-4.8
$ cd /usr/bin
$ sudo ln -s ./gcc-4.8 gcc
$ sudo ln -s ./g++-4.8 g++
```

To test your environment, install the packages used by the SDK examples (if you haven’t already) and then compile the examples:

```bash
$ sudo apt-get install libbz2-dev
$ sudo apt-get install libghc-zlib-dev
$ sudo apt-get install libboost-dev
$ sudo apt-get install libghc-curl-dev
$ cd /opt/vertica/sdk/examples
$ sudo make
```

Java Requirements

At a minimum, you need to install the following on your development machine:

- The Java Development Kit (JDK) version that matches the Java version you have installed on your database hosts (see Installing Java on Vertica Hosts).

- A copy of the Vertica SDK. See Setting Up the Java SDK for details.

While not required, make or some other build-management tool can ease development.

Python Requirements

Vertica does not require any additional files or packages. You can develop your Python UDx on any system with a text editor.
Important: When you begin UDx development, make sure that your code runs using Python 3.5.1. because you cannot change the version used by the Vertica Python interpreter.

When Vertica calls your UDx, it starts a side process that manages the interaction between the server and the Python interpreter.

R Requirements

Vertica does not require any additional files or packages. You can develop your R UDx on any system with a text editor.

Downloading and Running UDx Example Code

You can download all of the examples shown in this documentation, and many more, from the Vertica GitHub repository. This repository includes examples of all types of UDxs.

You can download the examples in either of two ways:

- Download the ZIP file. Extract the contents of the file into a directory.
- Clone the repository. Using a terminal window, run the following command:

  `git clone https://github.com/vertica/UDx-Examples.git`

The repository includes a makefile that you can use to compile the C++ and Java examples. It also includes .sql files that load and use the examples. See the README file for instructions on compiling and running the examples.

Running the examples not only helps you understand how a UDx works, but also helps you ensure your development environment is properly set up to compile UDx libraries.

See Also

- Aggregate Functions (UDAFs)
- Analytic Functions (UDAnFs)
- Scalar Functions (UDSFs)
Types of UDxs

Vertica supports five types of user-defined extensions:

- **User-Defined Aggregate Functions (UDAF)** allow you to create custom Aggregate Functions specific to your needs. They read one column of data, and return one output column. UDAFs can be developed in C++. Developing UDAFs is described in Aggregate Functions (UDAFs).

- **User-Defined Analytic Functions (UDAnF)** are similar to UDSFs, in that they read a row of data and return a single row. However, the function can read input rows independently of outputting rows, so that the output values can be calculated over several input rows. UDAnFs can be developed in C++ and Java. Developing UDAnFs is described in Analytic Functions (UDAnFs).

- **User-Defined Scalar Functions (UDSFs)** take in a single row of data and return a single value. These functions can be used anywhere a native function can be used, except CREATE TABLE BY PARTITION and SEGMENTED BY expressions. UDSFs can be developed in C++, Python, Java, and R. Developing UDSFs is described in Scalar Functions (UDSFs).

- **User-Defined Transform Functions (UDTFs)** operate on table segments and return zero or more rows of data. The data they return can be an entirely new table, unrelated to the schema of the input table, including having its own ordering and segmentation expressions. They can only be used in the SELECT list of a query. UDTFs can be developed in C++, Java, and R. Developing UDTFs is described in Transform Functions (UDTFs).

  To optimize query performance, you can use live aggregate projections to pre-aggregate the data that a UDTF returns. For more information, see Pre-Aggregating UDTF Results.

- **User-Defined Load (UDLs)** allows you to create custom sources, filters, and parsers to load data. These extensions are used in the COPY statement. UDLs can be developed in C++ and Java. Developing UDLs is described in Load (UDLs).

While each UDx type has a unique base class, developing them is similar in many ways. Different UDx types can also share the same library.
Handling Different Numbers and Types of Arguments

Usually, your UDxs accept a set number of arguments that are a specific data type (called its signature). You can create UDxs that handle multiple signatures, or even accept all arguments supplied to them by the user, using either of these techniques:

- Overloading your UDx by assigning the same SQL function name to multiple factory classes, each of which defines a unique function signature. When a user uses the function name in a query, Vertica tries to match the signature of the function call to the signatures declared by the factory's getPrototype method. This is the best technique to use if your UDx needs to accept a few different signatures (for example, accepting two required and one optional argument).

- Creating a polymorphic UDx by using the special "Any" argument type that tells Vertica to send all arguments that the user supplies to your function. Your UDx decides whether it can handle the arguments or not.

Overloading Your UDx

You may want your UDx to accept several different signatures (sets of arguments). For example, you might want your UDx to accept:

- One or more optional arguments.
- One or more arguments that can be one of several data types.
- Completely distinct signatures (either all INTEGER or all VARCHAR, for example).

You can create a function with this behavior by creating several factory classes, each of which accepts a different signature (the number and data types of arguments). You can then associate a single SQL function name with all of them. You can use the same SQL function name to refer to multiple factory classes as long as the signature defined by each factory is unique. When a user calls your UDx, Vertica matches the number and types of arguments supplied by the user to the arguments accepted by each of your function's factory classes. If one matches, Vertica uses it to instantiate a function class to process the data.
Multiple factory classes can instantiate the same function class, so you can re-use one function class that is able to process multiple sets of arguments and then create factory classes for each of the function signatures. You can also create multiple function classes if you want.

See the C++ Example: Overloading Your UDx and Java Example: Overloading Your UDx examples.

C++ Example: Overloading Your UDx

The following example code demonstrates creating a User-Defined Scalar Function (UDSF) that adds two or three integers together. The Add2or3ints class is prepared to handle two or three arguments. The processBlock() function checks the number of arguments that have been passed to it, and adds all two or three of them together. It also exits with an error message if it has been called with less than 2 or more than 3 arguments. In theory, this should never happen, since Vertica only calls the UDSF if the user’s function call matches a signature on one of the factory classes you create for your function. In practice, it is a good idea to perform this sanity checking, in case your (or someone else’s) factory class inaccurately reports a set of arguments your function class cannot handle.

```cpp
#include "Vertica.h"
using namespace Vertica;
using namespace std;
// a ScalarFunction that accepts two or three integers and adds them together.
class Add2or3ints : public Vertica::ScalarFunction
{
public:
  virtual void processBlock(Vertica::ServerInterface &srvInterface,
     Vertica::BlockReader &arg_reader,
     Vertica::BlockWriter &res_writer)
  {
    const size_t numCols = arg_reader.getNumCols();

    // Ensure that only two or three parameters are passed in
    if (numCols < 2 || numCols > 3)
      vt_report_error(0, "Function only accept 2 or 3 arguments, "
        "but %zu provided", arg_reader.getNumCols());
    // Add two integers together
    do {
      const vint a = arg_reader.getIntRef(0);
      const vint b = arg_reader.getIntRef(1);
      vint c = 0;
      // Check for third argument, add it in if it exists.
      if (numCols == 3)
        c = arg_reader.getIntRef(2);
      res_writer.setInt(a+b+c);
      res_writer.next();
    } while (arg_reader.next());
  }
  // This factory accepts function calls with two integer arguments.
};
```
class Add2intsFactory : public Vertica::ScalarFunctionFactory
{
  virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
  { return vt_createFuncObj(srvInterfaceallocator, Add2or3ints); }
  virtual void getPrototype(Vertica::ServerInterface &srvInterface,
                           Vertica::ColumnTypes &argTypes,
                           Vertica::ColumnTypes &returnType)
  {
    // Accept 2 integer values
    argTypes.addInt();
    argTypes.addInt();
    returnType.addInt();
  }
};
RegisterFactory(Add2intsFactory);
// This factory defines a function that accepts 3 ints.
class Add3intsFactory : public Vertica::ScalarFunctionFactory
{
  virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
  { return vt_createFuncObj(srvInterfaceallocator, Add2or3ints); }
  virtual void getPrototype(Vertica::ServerInterface &srvInterface,
                           Vertica::ColumnTypes &argTypes,
                           Vertica::ColumnTypes &returnType)
  {
    // accept 3 integer values
    argTypes.addInt();
    argTypes.addInt();
    argTypes.addInt();
    returnType.addInt();
  }
};
RegisterFactory(Add3intsFactory);

The example has two ScalarFunctionFactory classes, one for each signature that the function accepts (two integers and three integers). There is nothing unusual about these factory classes, except that their implementation of 
ScalarFunctionFactory::createScalarFunction() both create Add2or3ints objects.

The final step is to bind the same SQL function name to both factory classes. You can assign multiple factories to the same SQL function, as long as the signatures defined by each factory's 
getPrototype() implementation are different.

```sql
=> CREATE LIBRARY add2or3IntsLib AS '/home/dbadmin/Add2or3Ints.so';
CREATE LIBRARY
=> CREATE FUNCTION add2or3Ints as NAME 'Add2intsFactory' LIBRARY add2or3IntsLib FENCED;
CREATE FUNCTION
=> CREATE FUNCTION add2or3Ints as NAME 'Add3intsFactory' LIBRARY add2or3IntsLib FENCED;
CREATE FUNCTION
=> SELECT add2or3Ints(1,2);
add2or3Ints
-------------
 3
(1 row)
=> SELECT add2or3Ints(1,2,4);
```
add2or3Ints
-----------
    7
(1 row)
=> SELECT add2or3Ints(1,2,3,4); -- Will generate an error
ERROR 3467: Function add2or3Ints(int, int, int, int) does not exist, or
permission is denied for add2or3Ints(int, int, int, int)
HINT: No function matches the given name and argument types. You may
need to add explicit type casts

The error message in response to the final call to the add2or3Ints function was generated by
Vertica, since it could not find a factory class associated with add2or3Ints that accepted four
integer arguments. To expand add2or3Ints further, you could create another factory class that
accepted this signature, and either change the Add2or3Ints ScalarFunction class or create a
totally different class to handle adding more integers together. However, adding more classes
to accept each variation in the arguments quickly becomes overwhelming. In that case, you
should consider creating a polymorphic UDx.

Java Example: Overloading Your UDx

The following example code demonstrates creating a User-Defined Scalar Function (UDSF) that
adds two or three integers together. The Add2or3Ints class is prepared to handle two or three
arguments. It checks the number of arguments that have been passed to it, and adds all two or
three of them together. The `processBlock()` method checks whether it has been called with
less than 2 or more than 3 arguments. In theory, this should never happen, since Vertica only
calls the UDSF if the user's function call matches a signature on one of the factory classes you
create for your function. In practice, it is a good idea to perform this sanity checking, in case
your (or someone else's) factory class reports that your function class accepts a set of
arguments that it actually does not.

```java
// You need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.multiparamexample;
// Import the entire Vertica SDK
import com.vertica.sdk.*;
// This ScalarFunction accepts two or three integer arguments. It tests
// the number of input columns to determine whether to read two or three
// arguments as input.
public class Add2or3Ints extends ScalarFunction
{
    @Override
    public void processBlock(ServerInterface srvInterface,
                              BlockReader argReader,
                              BlockWriter resWriter)
            throws UdfException, DestroyInvocation
    {
        // See how many arguments were passed in
        int numCols = argReader.getNumCols();
    }
```
// Return an error if less than two or more than 3 arguments were given. This error only occurs if a factory class that accepts the wrong number of arguments instantiates this class.
if (numCols < 2 || numCols > 3) {
    throw new UdfException(0,
            "Must supply 2 or 3 integer arguments");
}

// Process all of the rows of input.
do {
    // Get the first two integer arguments from the BlockReader
    long a = argReader.getLong(0);
    long b = argReader.getLong(1);

    // Assume no third argument.
    long c = 0;

    // Get third argument value if it exists
    if (numCols == 3) {
        c = argReader.getLong(2);
    }

    // Process the arguments and come up with a result. For this example, just add the three arguments together.
    long result = a+b+c;

    // Write the integer output value.
    resWriter.setLong(result);

    // Advance the output BlockWriter to the next row.
    resWriter.next();

    // Continue processing input rows until there are no more.
} while (argReader.next());

The main difference between the Add2ints class and the Add2or3ints class is the inclusion of a section that gets the number of arguments by calling BlockReader.getNumCols(). This class also tests the number of columns it received from Vertica to ensure it is in the range it is prepared to handle. This test will only fail if you create a ScalarFunctionFactory whose getPrototype() method defines a signature that accepts less than two or more than three arguments. This is not really necessary in this simple example, but for a more complicated class it is a good idea to test the number of columns and data types that Vertica passed your function class.

Within the do loop, Add2or3ints uses a default value of zero if Vertica sent it two input columns. Otherwise, it retrieves the third value and adds that to the other two. Your own class needs to use default values for missing input columns or alter its processing in some other way to handle the variable columns.
You must define your function class in its own source file, rather than as an inner class of one of your factory classes since Java does not allow the instantiation of an inner class from outside its containing class. You factory class has to be available for instantiation by multiple factory classes.

Once you have created a function class or classes, you create a factory class for each signature you want your function class to handle. These factory classes can call individual function classes, or they can all call the same class that is prepared to accept multiple sets of arguments.

The following example's createScalarFunction() method instantiates a member of the Add2or3ints class.

```java
package com.mycompany.multiparamexample;

import com.vertica.sdk.*;

public class Add2or3intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
                               ColumnTypes argTypes,
                               ColumnTypes returnType)
    {
        // Accept two integers as input
        argTypes.addInt();
        argTypes.addInt();
        // writes one integer as output
        returnType.addInt();
    }
    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface)
    {
        // Instantiate the class that can handle either 2 or 3 integers.
        return new Add2or3ints();
    }
}
```

The following ScalarFunctionFactory subclass accepts three integers as input. It, too, instantiates a member of the Add2or3ints class to process the function call:

```java
package com.mycompany.multiparamexample;

import com.vertica.sdk.*;

public class Add3intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
                               ColumnTypes argTypes,
                               ColumnTypes returnType)
    {
        // Accepts three integers as input
        argTypes.addInt();
    }
}
```
argTypes.addInt();
argTypes.addInt();
// Returns a single integer
returnType.addInt();
}

@Override
public ScalarFunction createScalarFunction(ServerInterface srvInterface)
{
// Instantiates the Add2or3ints ScalarFunction class, which is able to
// handle either 2 or 3 integers as arguments.
    return new Add2or3ints();
}

The factory classes and the function class or classes they call must be packaged into the same JAR file (see Compiling and Packaging a Java Library for details). If a host in the database cluster has the JDK installed on it, you could use the following commands to compile and package the example:

$ cd pathToJavaProject$
$ javac -classpath /opt/vertica/bin/VerticaSDK.jar \
> com/mycompany/multiparamexample/*.java
$ jar -cvf Add2or3intslib.jar com/vertica/sdk/BuildInfo.class \
> com/mycompany/multiparamexample/*.class

Once you have packaged your overloaded UDX, you deploy it the same way as you do a regular UDX, except you use multiple CREATE FUNCTION statements to define the function, once for each factory class.

=> CREATE LIBRARY add2or3intslib as '*/home/dbadmin/Add2or3intslib.jar'
=> language 'Java';
CREATE LIBRARY
=> CREATE FUNCTION add2or3ints as LANGUAGE 'Java' NAME
'com.mycompany.multiparamexample/Add2intsFactory' LIBRARY add2or3intslib;
CREATE FUNCTION
=> CREATE FUNCTION add2or3ints as LANGUAGE 'Java' NAME
'com.mycompany.multiparamexample/Add3intsFactory' LIBRARY add2or3intslib;
CREATE FUNCTION

You call the overloaded function the same way you call any other function.

=> SELECT add2or3ints(2,3);
    add2or3ints
    -----------
    5
(1 row)
=> SELECT add2or3ints(2,3,4);
    add2or3ints
    -----------
    9
The last error was generated by Vertica, not the UDx code. It returns an error if it cannot find a factory class whose signature matches the function call's signature.

Creating an overloaded UDx is useful if you want your function to accept a limited set of potential arguments. If you want to create a more flexible function, you can create a polymorphic function.

Creating a Polymorphic UDx

Polymorphic UDxs accept any number and type of argument that the user supplies. Vertica does not check the number or types of argument that the user passes to the UDx—it just passes the UDx all of the arguments supplied by the user. It is up to your polymorphic UDx's main processing function (for example, processBlock() in User-Defined Scalar Functions) to examine the number and types of arguments it received and determine if it can handle them.

Note: UDxs have a maximum 1600 arguments.

Polymorphic UDxs are more flexible than using multiple factory classes for your function (see Overloading Your UDx), because your function can determine at run time if it can process the arguments rather than accepting specific sets of arguments. However, your polymorphic function needs to perform more work to determine whether it can process the arguments that it has been given.

Your polymorphic UDx declares that it accepts any number of arguments in its factory's getPrototype() function by calling the addAny() function on the ColumnTypes object that defines its arguments. This "any parameter" argument type is the only one that your function can declare. You cannot define required arguments and then call addAny() to declare the rest of the signature as optional. If your function has requirements for the arguments it accepts, your process function must enforce them.

Polymorphic UDxs and Schema Search Paths

If a user does not supply a schema name as part of a UDx call, Vertica searches each schema in the schema search path for a function whose name and signature match the function call. See Setting Search Paths in the Administrator's Guide for more information about schema search paths.
Since polymorphic UDxs do not have a specific signature associated with them, Vertica initially skips them when searching for a function to handle the function call. If none of the schemas in the search path contain a U Dx whose name and signature match the function call, Vertica searches the schema search path again for a polymorphic UDx whose name matches the function name in the function call.

This behavior gives precedence to UDxs whose signature exactly matches the function call. It allows you to create a "catch all" polymorphic UDx that Vertica calls only when none of the non-polymorphic UDxs with the same name have matching signatures.

This behavior may cause confusion if your users expect the first polymorphic function in the schema search path to handle a function call. To avoid confusion, you should:

- Avoid using the same name for different UDxs. You should always uniquely name UDxs unless you intend to create an overloaded UDx with multiple signatures.

- When you cannot avoid having UDxs with the same name in different schemas, always supply the schema name as part of the function call. Using the schema name prevents ambiguity and ensures that Vertica uses the correct UDx to process your function calls.

### C++ Example: Polymorphic UDx

The following example shows an implementation of a ScalarFunction that adds together two or more integers.

```c++
#include "Vertica.h"
using namespace Vertica;
using namespace std;

// Adds two or more integers together.
class AddManyInts : public Vertica::ScalarFunction
{
public:
    virtual void processBlock(Vertica::ServerInterface &srvInterface,
                          Vertica::BlockReader &arg_reader,
                          Vertica::BlockWriter &res_writer)
    {
        // Always catch exceptions to prevent causing the side process or
        // Vertica itself from crashing.
        try
        {
            // Find the number of arguments sent.
            size_t numCols = arg_reader.getNumCols();

            // Make sure at least 2 arguments were supplied
            if (numCols < 2)
                vt_report_error(0, "Function expects at least 2 integer parameters");

            // Make sure all types are ints
            const SizedColumnTypes &inTypes = arg_reader.getTypeMetaData();
            for (int param=0; param < (int)numCols; param++)
```
Most of the work in the example is done by the `ScalarFunction.processBlock()` function. It performs two checks on the arguments that have been passed in through the `BlockReader` object:

- **Ensures there are at least two arguments**
- **Checks the data type of all arguments to ensure they are all integers**

Once the checks are performed, the example processes the block of data by looping over the arguments and adding them together.
You assign a SQL name to your polymorphic UDx using the same statement you use to assign one to a non-polymorphic UDx. The following demonstration shows how you load and call the polymorphic function from the example.

```sql
CREATE LIBRARY addManyIntsLib AS '/home/dbadmin/AddManyInts.so';
CREATE FUNCTION addManyInts AS NAME 'AddManyIntsFactory' LIBRARY addManyIntsLib FENCED;
CREATE FUNCTION
SELECT addManyInts(1,2);
addManyInts 3
(1 row)
SELECT addManyInts(1,2,3,40,60,70,80,900);
addManyInts 1206
(1 row)
SELECT addManyInts(1); -- Too few parameters
ERROR 3412: Failure in UDx RPC call InvokeProcessBlock(): Error calling processBlock() in User-Defined Object [addManyInts] at [AddManyInts.cpp:51], error code: 0, message: Exception while processing partition: [Function expects at least 2 integer parameters]
SELECT addManyInts(1,2.232343); -- Wrong data type
ERROR 3412: Failure in UDx RPC call InvokeProcessBlock(): Error calling processBlock() in User-Defined Object [addManyInts] at [AddManyInts.cpp:51], error code: 0, message: Exception while processing partition: [Function expects all arguments to be INTEGER. Argument 2 was Numeric(7,6)]
```

Notice that the errors returned by last two calls to the function were generated by the `processBlock()` function. It is up to your UDx to ensure that the user supplies the correct number and types of arguments to your function and exit with an error if it cannot process them.

**Java Example: Polymorphic UDx**

The following example shows an implementation of a `ScalarFunctionFactory` class with an inner `ScalarFunction` class that adds together two or more integers.

```java
public class AddManyIntsFactory extends ScalarFunctionFactory {
    @Override
    public void getPrototype(ServerInterface srvInterface,
                              ColumnTypes argTypes,
                              ColumnTypes returnType)
```

Vertica Analytic Database (9.0.x)  Page 4160 of 6180
{  
    // Accepts any number and type or arguments. The ScalarFunction  
    // class handles parsing the arguments.  
    argTypes.addAny();  
    // writes one integer as output  
    returnType.addInt();  
}  
// This polymorphic ScalarFunction adds all of the integer arguments passed  
// to it. Returns an error if there are less than two arguments, or if one  
// argument is not an integer.  
public class AddManyInts extends ScalarFunction  
{  
    @Override  
    public void processBlock(ServerInterface srvInterface,  
              BlockReader argReader,  
              BlockWriter resWriter)  
        throws UdfException, DestroyInvocation  
    {  
        // See how many arguments were passed in  
        int numCols = argReader.getNumCols();  
        
        // Return an error if less than two arguments were given.  
        if (numCols < 2) {  
            throw new UdfException(0,  
                       "Must supply at least 2 integer arguments");  
        }  
        
        // Make sure all input columns are integer.  
        SizedColumnTypes inTypes = argReader.getTypeMetaData();  
        for (int param = 0; param < numCols; param++) {  
            VerticaType paramType = inTypes.getColumnType(param);  
            if (!paramType.isInt()) {  
                throw new UdfException(0, "Error: Argument "+ (param+1) +  
                                       " was not an integer. All arguments must be integer.");  
            }  
        }  
        
        // Process all of the rows of input.  
        do {  
            long total = 0; // Hold the running total of arguments  
            
            // Get all of the arguments and add them up  
            for (int x = 0; x < numCols; x++) {  
                total += argReader.getLong(x);  
            }  
            
            // Write the integer output value.  
            resWriter.setLong(total);  
            
            // Advance the output BlockWriter to the next row.  
            resWriter.next();  
            
            // Continue processing input rows until there are no more.  
            } while (argReader.next());  
    }  
}  

@Override  
public ScalarFunction createScalarFunction(ServerInterface srvInterface)  
{
// Instantiate the polymorphic UDF class.
return new AddManyInts();
}

The ScalarFunctionFactory.getPrototype() method calls the addAny() method to declare that the UDSF is polymorphic.

Most of the work in the example is done by the ScalarFunction.processBlock() method. It performs two checks on the arguments that have been passed in through the BlockReader object:

- There are at least two arguments.
- The data type of all arguments are integers.

It is up to your polymorphic UDX to determine that all of the input passed to it is valid.

Once the processBlock() method validates its arguments, it loops over them, adding them together.

You assign a SQL name to your polymorphic UDX using the same statement you use to assign one to a non-polymorphic UDX. The following demonstration shows how you load and call the polymorphic function from the example.

```
=> CREATE LIBRARY addmanyintslib AS '/home/dbadmin/AddManyIntsLib.jar'
  LANGUAGE 'Java';
CREATE LIBRARY => create function addmanyints AS LANGUAGE 'Java' NAME => 'com.mycompany.multiparamexample.AddManyIntsFactory' LIBRARY addmanyintslib;
CREATE FUNCTION => SELECT addmanyints(1,2,3,4,5,6,7,8,9,10);
addmanyints
------------------
  55
(1 row)
=> SELECT addmanyints(1); --Too few parameters
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User Defined Object [addmanyints], error code: 0
com.vertica.sdk.UdfException: Must supply at least 2 integer arguments at com.mycompany.multiparamexample.AddManyIntsFactory$AddManyInts.processBlock
(AddManyIntsFactory.java:39)
  at com.vertica.udxfence.UdxExecContext.processBlock(UdxExecContext.java:700)
  at com.vertica.udxfence.UdxExecContext.run(UdxExecContext.java:173)
  at java.lang.Thread.run(Thread.java:662)
=> SELECT addmanyints(1,2,3,14159); --Non-integer parameter
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User Defined Object [addmanyints], error code: 0
com.vertica.sdk.UdfException: Error: Argument 3 was not an integer. All arguments must be integer.
at com.mycompany.multiparamexample.AddManyIntsFactory$AddManyInts.processBlock
(AddManyIntsFactory.java:48)
```
R Example: Polymorphic UDx

The following example shows an implementation of a Transform Function (UDTF) that performs kmeans clustering on one or more input columns.

```r
kmeansPoly <- function(v.data.frame, v.param.list) {
  # Computes clusters using the kmeans algorithm.
  # # Input: A dataframe and a list of parameters.
  # # Output: A dataframe with one column that tells the cluster to which each data
  # # point belongs.
  # Args:
  # v.data.frame: The data from Vertica cast as an R data frame.
  # v.param.list: List of function parameters.
  # # Returns: # The cluster associated with each data point.
  # Ensure k is not null.
  if(!is.null(v.param.list[['k']])) {
    number_of_clusters <- as.numeric(v.param.list[['k']])
  } else {
    stop("k cannot be NULL! Please use a valid value.")
  }
  # Run the kmeans algorithm.
  kmeans_clusters <- kmeans(v.data.frame, number_of_clusters)
  final.output <- data.frame(kmeans_clusters$cluster)
  return(final.output)
}

kmeansFactoryPoly <- function() {
  # This function tells Vertica the name of the R function,
  # and the polymorphic parameters.
  list(name=kmeansPoly, udxtype=c("transform"), intype=c("any"),
       outtype=c("int"), parametertypecallback=kmeansParameters)
}

kmeansParameters <- function() {
  # Callback function for the parameter types.
  function.parameters <- data.frame(datatype=rep(NA, 1), length=rep(NA, 1),
                                     scale=rep(NA, 1), name=rep(NA, 1))
  function.parameters[1,1] = "int"
  function.parameters[1,4] = "k"
  return(function.parameters)
}
```

The polymorphic R function declares it accepts any number of arguments in its factory function by specifying "any" as the argument to the intype parameter and optionally the outtype parameter. If you define "any" argument for intype or outtype, then it is the only type that your function can declare for the respective parameter. You cannot define required arguments...
and then call "any" to declare the rest of the signature as optional. If your function has requirements for the arguments it accepts, your process function must enforce them.

The `outtypecallback` method is used to indicate the argument types and sizes it has been called with, and is expected to indicate the types and sizes that the function returns. The `outtypecallback` method can also be used to check for unsupported types and/or number of arguments. For example, the function may require only integers, with no more than 10 of them.

You assign a SQL name to your polymorphic U Dx using the same statement you use to assign one to a non-polymorphic U Dx. The following statements show how you load and call the polymorphic function from the example.

```sql
=> CREATE LIBRARY rlib2 AS '/home/dbadmin/R_UDx/poly_kmeans.R' LANGUAGE 'R';
CREATE LIBRARY
=> CREATE TRANSFORM FUNCTION kmeansPoly AS LANGUAGE 'R' name 'kmeansFactoryPoly' LIBRARY rlib2;
CREATE FUNCTION
=> SELECT spec, kmeansPoly(sl,sw,pl,pw USING PARAMETERS k = 3)
       OVER(PARTITION BY spec) AS Clusters
       FROM iris;
spec | Clusters
---+----------------
Iris-setosa | 1
Iris-setosa | 1
Iris-setosa | 1
Iris-setosa | 1
... ...
(150 rows)
```
### UDx Parameters

Parameters let you define arguments for your UDxs that remain constant across all of the rows processed by the SQL statement that calls your UDx. Typically, your UDxs accept arguments that come from columns in a SQL statement. For example, in the following SQL statement, the arguments `a` and `b` to the `add2ints` UDFS change value for each row processed by the SELECT statement:

```sql
=> SELECT a, b, add2ints(a,b) AS 'sum' FROM example;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>19</td>
</tr>
</tbody>
</table>

(5 rows)

Parameters remain constant for all the rows your UD processes. You can also make parameters optional so that if the user does not supply it, your UD uses a default value. For example, the following example demonstrates calling a UDFS named `add2intsWithConstant` that has a single parameter value named `constant` whose value is added to each the arguments supplied in each row of input:

```sql
=> SELECT a, b, add2intsWithConstant(a, b USING PARAMETERS constant=42) AS 'a+b+42' FROM example;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a+b+42</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>61</td>
</tr>
</tbody>
</table>

(5 rows)

**Note:** When calling a UDx with parameters, there is no comma between the last argument and the USING PARAMETERS clause.

The topics in this section explain how to develop UDxs that accept parameters.

### Defining the Parameters Your UDx Accepts

You define the parameters that your UDx accepts in its factory class (`ScalarFunctionFactory`, `AggregateFunctionFactory`, and so on) by implementing
getParameterType(). This method is similar to getReturnType(): you call data-type-specific methods on a SizedColumnTypes object that is passed in as a parameter. Each function call sets the name, data type, and width or precision (if the data type requires it) of the parameter.

### Setting Parameter Properties (C++ Only)

When you add parameters to the getParameterType() function using the C++ API, you can also set properties for each parameter. For example, you can define a parameter as being required by the UDx. Doing so lets the Vertica server know that every UDx invocation must provide the specified parameter, or the query fails.

By passing an object to the SizedColumnTypes:Properties class, you can define the following four parameter properties:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>BOOLEAN</td>
<td>If set to TRUE, the parameter appears in the USER_FUNCTION_PARAMETERS table. You may want to set this to FALSE to declare a parameter for internal use only.</td>
</tr>
<tr>
<td>required</td>
<td>BOOLEAN</td>
<td>If set to TRUE:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The parameter is required when invoking the UDx.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Invoking the UDx without supplying the parameter results in an error, and the UDx does not run.</td>
</tr>
<tr>
<td>canBeNull</td>
<td>BOOLEAN</td>
<td>If set to TRUE, the parameter can have a NULL value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If set to FALSE, make sure that the supplied parameter does not contain a NULL value when invoking the UDx. Otherwise, an error results, and the UDx does not run.</td>
</tr>
<tr>
<td>comment</td>
<td>VARCHAR (128)</td>
<td>A comment to describe the parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you exceed the 128 character limit, Vertica generates an error when you run the CREATE_FUNCTION command. Additionally, if you replace the existing function definition in the comment parameter, make sure that the new definition does not exceed 128 characters. Otherwise, you delete all existing entries in the USER_FUNCTION_PARAMETERS table related to the UDx.</td>
</tr>
</tbody>
</table>
Setting Parameter Properties (R Only)

When using parameters in your R UDx, you must specify a field in the factory function called `parametertypecallback`. This field points to the callback function that defines the parameters expected by the function. The callback function defines a four-column data frame with the following properties:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype</td>
<td>VARCHAR(128)</td>
<td>The data type of the parameter.</td>
</tr>
<tr>
<td>length</td>
<td>INTEGER</td>
<td>The dimension of the parameter.</td>
</tr>
<tr>
<td>scale</td>
<td>INTEGER</td>
<td>The proportional dimensions of the parameter.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the parameter.</td>
</tr>
</tbody>
</table>

If any of the columns are left blank (or the `parametertypecallback` function is omitted), then Vertica uses default values.

For more information, see `parametertypecallback Function`.

Getting Parameter Values in UDxs

Your UDx uses the parameter values it declared in its factory class (see `Defining the Parameters Your UDx Accepts`) in its function class's processing method (for example, `processBlock()` or `processPartition()`). It gets its parameter values from a `ParamReader` object, which is available from the `ServerInterface` object that is passed to your processing method. Reading parameters from this object is similar to reading argument values from `BlockReader` or `PartitionReader` objects: you call a data-type-specific function with the name of the parameter to retrieve its value. For example, in C++:

```cpp
// Get the parameter reader from the ServerInterface to see if there are supplied parameters.
ParamReader paramReader = srvInterface.getParamReader();
// Get the value of an int parameter named constant.
const vint constant = paramReader.getIntRef("constant");
```

**Note:** String data values do not have any of their escape characters processed before they are passed to your function. Therefore, your function may need to process the escape sequences itself if it needs to operate on unescaped character values.
Using Parameters in the Factory Class

In addition to using parameters in your UDx function class, you can also access the parameters in the factory class. You may want to access the parameters to let the user control the input or output values of your function in some way. For example, your UDx can have a parameter that lets the user choose to have your UDx return a single- or double-precision value. The process of accessing parameters in the factory class is the same as accessing it in the function class: get a ParamReader object from the ServerInterface's getParamReader() method, then read the parameter values.

Testing Whether the User Supplied Parameter Values

Unlike its handling of arguments, Vertica does not immediately return an error if a user's function call does not include a value for a parameter defined by your UDx's factory class. This means that your function can attempt to read a parameter value that the user did not supply. If it does so, by default Vertica returns a non-existent parameter warning to the user, and the query containing the function call continues.

If you want your parameter to be optional, you can test whether the user supplied a value for the parameter before attempting to access its value. Your function determines if a value exists for a particular parameter by calling the ParamReader's containsParameter() method with the parameter's name. If this call returns true, your function can safely retrieve the value. If this call returns false, your UDx can use a default value or change its processing in some other way to compensate for not having the parameter value. As long as your UDx does not try to access the non-existent parameter value, Vertica does not generate an error or warning about missing parameters.

Note: If the user passes your UDx a parameter that it has not defined, by default Vertica issues a warning that the parameter is not used. It still executes the SQL statement, ignoring the parameter. You can change this behavior by altering the StrictUDxParameterChecking configuration parameter.

See C++ Example: Defining Parameters for an example.
Calling UDxs with Parameters

You pass parameters to a UDx by adding a USING PARAMETERS clause in the function call after the last argument.

- Do not insert a comma between the last argument and the USING PARAMETERS clause.
- After the USING PARAMETERS clause, add one or more parameter definitions, in the following form:

  `<parameter name> = <parameter value>`

  - Separate parameter definitions by commas.

Parameter values can be a constant expression (for example `1234 + SQRT(5678)`). You cannot use volatile functions (such as `RANDOM`) in the expression, because they do not return a constant value. If you do supply a volatile expression as a parameter value, by default, Vertica returns an incorrect parameter type warning. Vertica then tries to run the UDx without the parameter value. If the UDx requires the parameter, it returns its own error, which cancels the query.

Calling a UDx with a Single Parameter

The following example demonstrates how you can call the Add2intsWithConstant UDSF example shown in C++ Example: Defining Parameters:

```sql
=> SELECT a, b, Add2intsWithConstant(a, b USING PARAMETERS constant=42) AS 'a+b+42' from example;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a+b+42</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>61</td>
</tr>
</tbody>
</table>

(5 rows)

To remove the first instance of the number 3, you can call the RemoveSymbol UDSF example:

```sql
=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS symbol='3');
```

<table>
<thead>
<tr>
<th>original_string</th>
<th>RemoveSymbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>3re3mo3ve3sy3mb3ol</td>
<td>re3mo3ve3sy3mb3ol</td>
</tr>
</tbody>
</table>

(1 row)
Calling a UDx with Multiple Parameters

The following example shows how you can call a version of the tokenize UDTF. This UDTF includes parameters to limit the shortest allowed word and force the words to be output in uppercase. Separate multiple parameters with commas.

```sql
=> SELECT url, tokenize(description USING PARAMETERS minLength=4, uppercase=true) OVER (partition by url) FROM T;
```

<table>
<thead>
<tr>
<th>url</th>
<th>words</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>ONLINE</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>RETAIL</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>MERCHAND</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>PROVIDER</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>CLOUD</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>SERVICES</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>LEADING</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>PROVIDER</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>COMPUTER</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>HARDWARE</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>IMAGING</td>
</tr>
<tr>
<td><a href="http://www.hpe.com">www.hpe.com</a></td>
<td>SOLUTIONS</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>WORLD'S</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>FASTEST</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>ANALYTIC</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>DATABASE</td>
</tr>
</tbody>
</table>

(16 rows)

The following example calls the RemoveSymbol UDSF. By changing the value of the optional parameter, n, you can remove all instances of the number 3:

```sql
=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS symbol='3', n=6);
```

<table>
<thead>
<tr>
<th>original_string</th>
<th>RemoveSymbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>3re3mo3ve3sy3mb3ol</td>
<td>removesymbol</td>
</tr>
</tbody>
</table>

(1 row)

Calling a UDx with Optional or Incorrect Parameters

You can optionally add the Add2intsWithConstant UDSF's constant parameter. Calling this constraint without the parameter does not return an error or warning:

```sql
=> SELECT a,b,Add2intsWithConstant(a, b) AS 'sum' FROM example;
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>

(16 rows)
Although calling a UDx with incorrect parameters generates a warning, by default, the query still runs. For further information on setting the behavior of your UDx when you supply incorrect parameters, see Specifying the Behavior of Passing Unregistered Parameters.

```sql
=> SELECT a, b, add2intsWithConstant(a, b USING PARAMETERS wrongparam=42) AS 'result' from example;

WARNING 4332: Parameter wrongparam was not registered by the function and cannot be coerced to a definite data type

    a | b | result
----------
    1 | 2 | 3
    3 | 4 | 7
    5 | 6 | 11
    7 | 8 | 15
    9 | 10| 19

(5 rows)
```

Specifying the Behavior of Passing Unregistered Parameters

By default, Vertica issues a warning message when you pass a UDx an unregistered parameter. An unregistered parameter is one that you did not declare in the `getParameterType()` method.

You can control the behavior of your UDx when you pass it an unregistered parameter by altering the `StrictUDxParameterChecking` configuration parameter.

Unregistered Parameter Behavior Settings

You can specify the behavior of your UDx in response to one or more unregistered parameters. To do so, set the `StrictUDxParameterChecking` configuration parameter to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Allows unregistered parameters to be accessible to the UDx. The <code>ParamReader</code> class's <code>getType()</code> method determines the data type of the unregistered parameter. Vertica does not display any warning or error message.</td>
</tr>
</tbody>
</table>
### Examples

The following examples demonstrate the behavior you can specify using different values with the StrictUDxParameterChecking parameter.

**View the Current Value of StrictUDxParameterChecking**

To view the current value of the StrictUDxParameterChecking configuration parameter, run the following query:

```sql
=> \x
Expanded display is on.
=> SELECT * FROM configuration_parameters WHERE parameter_name = 'StrictUDxParameterChecking';
- [ RECORD 1 ]-----------------------------------------------
    node_name | ALL
    parameter_name | StrictUDxParameterChecking
    current_value | 1
    restart_value | 1
    database_value | 1
    default_value | 1
    current_level | DATABASE
    restart_level | DATABASE
    is_mismatch | f
    groups | |
    allowed_levels | DATABASE
    superuser_only | f
    change_under_support_guidance | f
    change_requires_restart | f
    description | Sets the behavior to deal with undeclared UDx function parameters
```

**Change the Value of StrictUDxParameterChecking**

You can change the value of the StrictUDxParameterChecking configuration parameter at the database, node, or session level. For example, you can change the value to '0' to specify that unregistered parameters can pass to the UDx without displaying a warning or error message:

```sql
=> ALTER DATABASE mydb SET StrictUDxParameterChecking = 0;
```
Invalid Parameter Behavior with RemoveSymbol

The following example demonstrates how to call the RemoveSymbol UDSF example. The RemoveSymbol UDSF has a required parameter, symbol, and an optional parameter, n. In this case, you do not use the optional parameter.

If you pass both symbol and an additional parameter called wrongParam, which is not declared in the UDx, the behavior of the UDx changes corresponding to the value of StrictUDxParameterChecking.

When you set StrictUDxParameterChecking to '0', the UDx runs normally without a warning. Additionally, wrongParam becomes accessible to the UDx through the ParamReader object of the ServerInterface object:

```sql
=> ALTER DATABASE mydb SET StrictUDxParameterChecking = 0;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS symbol='3', wrongParam='x');
  original_string  | RemoveSymbol
-------------------+------------------
  3re3mo3ve3sy3mb3ol | re3mo3ve3sy3mb3ol
(1 row)
```

When you set StrictUDxParameterChecking to '1', the UDx ignores wrongParam and runs normally. However, it also issues a warning message:

```sql
=> ALTER DATABASE mydb SET StrictUDxParameterChecking = 1;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS symbol='3', wrongParam='x');
WARNING 4320: Parameter wrongParam was not registered by the function and cannot be coerced to a definite data type
  original_string  | RemoveSymbol
-------------------+------------------
  3re3mo3ve3sy3mb3ol | re3mo3ve3sy3mb3ol
(1 row)
```

When you set StrictUDxParameterChecking to '2', the UDx encounters an error when it tries to call wrongParam and does not run. Instead, it generates an error message:

```sql
=> ALTER DATABASE mydb SET StrictUDxParameterChecking = 2;
ALTER DATABASE

=> SELECT '3re3mo3ve3sy3mb3ol' original_string, RemoveSymbol('3re3mo3ve3sy3mb3ol' USING PARAMETERS symbol='3', wrongParam='x');
ERROR 0: Parameter wrongParam was not registered by the function
```
See Also

- Setting Configuration Parameter Values

User-Defined Session Parameters

User-defined session parameters allow you to write more generalized parameters than what Vertica provides. You can configure user-defined session parameters in these ways:

- From the client—for example, with `ALTER SESSION`
- Through the UDx itself

A user-defined session parameter can be passed into any type of UDx supported by Vertica. You can also set parameters for your UDx at the session level. By specifying a user-defined session parameter, you can have the state of a parameter saved continuously. Vertica saves the state of the parameter even when the UDx is invoked multiple times during a single session.

The RowCount example uses a user-defined session parameter. This parameter counts the total number of rows processed by the UDx each time it runs. RowCount then displays the aggregate number of rows processed for all executions. See C++ Example: Using Session Parameters and Java Example: Using Session Parameters for implementations.

Viewing the User-Defined Session Parameter

Enter the following command to see the value of the session parameter:

```sql
=> SHOW SESSION UDPARAMETER all;
schema | library | key | value
--------+--------+----+------
(0 rows)
```

Now, execute the UDx:

```sql
=> SELECT RowCount(5,5);
RowCount
--------
10
(1 row)
```

Again, enter the command to see the value of the session parameter:
Because the UDx has processed one row, the value of RowCount is now 1. Running the UDx two more times should increase the value of RowCount by 2:

```
=> SELECT RowCount(10,10);
RowCount
--------
20
(1 row)
=> SELECT RowCount(15,15);
RowCount
--------
30
(1 row)
```

You have now executed the UDx three times, obtaining the sum of 5 + 5, 10 + 10, and 15 + 15. Now, check the value of RowCount.

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key   | value
--------+---------+-------+-------
public | UDSession| rowcount | 3
(1 row)
```

### Altering the User-Defined Session Parameter

You can also manually alter the value of RowCount. To do so, enter the following command:

```
=> ALTER SESSION SET UDPARAMETER FOR UDSession RowCount = 25;
ALTER SESSION
```

Check the value of RowCount:

```
=> SHOW SESSION UDPARAMETER all;
schema | library | key   | value
--------+---------+-------+-------
public | UDSession| rowcount | 25
(1 row)
```

### Clearing the User-Defined Session Parameter

**From the client:**

To clear the current value of RowCount, enter the following command:
ALTER SESSION CLEAR UPARAMETER FOR UDSession RowCount;
ALTER SESSION

Verify that RowCount has been cleared:

SHOW SESSION UPARAMETER all;
schema | library | key | value
----------+---------+-----+-------
(0 rows)

Through the UDx in C++:

You can set the session parameter to clear through the UDx itself. For example, to clear RowCount when its value reaches 10 or greater do the following:

1. Remove the following line from the destroy() method in RowCount:

   udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);

2. Replace the removed line from the destroy() method with the following code:

   if (rowCount < 10)
   {
     udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);
   }
   else
   {
     udParams.getUDSessionParamWriter("library").clearParameter("rowCount");
   }

3. To see the UDx clear the session parameter, set RowCount to a value of 9:

   ALTER SESSION SET UPARAMETER FOR UDSession RowCount = 9;
   ALTER SESSION

4. Check the value of RowCount:

   SELECT RowCount(15,15);
   RowCount
   -------
   30
6. Check the value of RowCount. Because the value has reached 10, the threshold specified in the UDx, expect that RowCount is cleared:

```
=> SHOW SESSION UDPARAMETER all;
```

<table>
<thead>
<tr>
<th>schema</th>
<th>library</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>(0 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As expected, RowCount is cleared.

**Through the UDx in Java:**

1. Remove the following lines from the `destroy()` method in `RowCount`:

   ```java
   udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
   srvInterface.log("RowNumber processed %d records", count);
   ```

2. Replace the removed lines from the `destroy()` method with the following code:

   ```java
   if (rowCount < 10) {
   udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
   srvInterface.log("RowNumber processed %d records", count);
   } else {
   udParams.getUDSessionParamWriter("library").clearParameter("rowCount");
   }
   ```

3. To see the UDx clear the session parameter, set RowCount to a value of 9:

   ```
   => ALTER SESSION SET UDPARAMETER FOR UDSession RowCount = 9;
   ALTER SESSION
   ```

4. Check the value of RowCount:

   ```
   => SHOW SESSION UDPARAMETER all;
   ```

<table>
<thead>
<tr>
<th>schema</th>
<th>library</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>UDSession</td>
<td>rowcount</td>
<td>9</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Invoke RowCount so that its value becomes 10:
SELECT RowCount(15,15);
RowCount
----------
   30
(1 row)

6. Check the value of RowCount. Since the value has reached 10, the threshold specified in the UDx, expect that RowCount is cleared:

SHOW SESSION UDPARAMETER all;
schema | library | key    | value
----------+---------+--------+--------
          |         |        |        
(0 rows)

As expected, RowCount is cleared.

Read-Only and Hidden Session Parameters

If you don't want a parameter to be set anywhere except in the UDx, you can make it read-only. If, additionally, you don't want a parameter to be visible in the client, you can make it hidden.

To make a parameter read-only, meaning that it cannot be set in the client, but can be viewed, add a single underscore before the parameter's name. For example, to make rowCount read-only, change all instances in the UDx of "rowCount" to "_rowCount".

To make a parameter hidden, meaning that it cannot be viewed in the client nor set, add two underscores before the parameter's name. For example, to make rowCount hidden, change all instances in the UDx of "rowCount" to "__rowCount".

See Also

User-Defined Session Parameters

C++ Example: Defining Parameters

The following code fragment demonstrates adding a single parameter to the C++ add2ints UDSF example. The getParameterType() function defines a single integer parameter that is named constant.

class Add2intsWithConstantFactory : public ScalarFunctionFactory
{

```cpp
```
// Return an instance of Add2ints to perform the actual addition.
virtual ScalarFunction *createScalarFunction(ServerInterface &interface)
{
    // Calls the vt_createFuncObj to create the new Add2ints class instance.
    return vt_createFuncObj(interface.allocator, Add2intsWithConstant);
}

// Report the argument and return types to Vertica.
virtual void getPrototype(ServerInterface &interface,
    ColumnTypes &argTypes,
    ColumnTypes &returnType)
{
    // Takes two ints as inputs, so add ints to the argTypes object.
    argTypes.addInt();
    argTypes.addInt();
    // Returns a single int.
    returnType.addInt();
}

// Defines the parameters for this UDSF. Works similarly to defining arguments and return types.
virtual void getParameterType(ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes)
{
    // One int parameter named constant.
    parameterTypes.addInt("constant");
}

};
RegisterFactory(Add2intsWithConstantFactory);

See the Vertica SDK entry for SizedColumnType for a full list of the data-type-specific functions you can call to define parameters.

The following code fragment demonstrates using the parameter value. The Add2intsWithConstant class defines a function that adds two integer values. If the user supplies it, the function also adds the value of the optional integer parameter named constant.

/***
 * A UDSF that adds two numbers together with a constant value.
 *
 */
class Add2intsWithConstant : public ScalarFunction
{
public:
    // Processes a block of data sent by Vertica.
    virtual void processBlock(ServerInterface &srvInterface,
        BlockReader &arg_reader,
        BlockWriter &res_writer)
    {
        try
        {
            // The default value for the constant parameter is 0.
            vint constant = 0;
            // Get the parameter reader from the ServerInterface to see if there are supplied parameters.
            ParamReader paramReader = srvInterface.getParamReader();
            // See if the user supplied the constant parameter.
            if (paramReader.containsParameter("constant"))
                // There is a parameter, so get its value.

C++ Example: Using Session Parameters

The RowCount example uses a user-defined session parameter, also called RowCount. This parameter counts the total number of rows processed by the UDX each time it runs. RowCount then displays the aggregate number of rows processed for all executions.

```cpp
#include <string>
#include <sstream>
#include <iostream>
#include "Vertica.h"
#include "VerticaUDx.h"

using namespace Vertica;

class RowCount : public Vertica::ScalarFunction
{
private:
    int rowCount;
    int count;

public:
    virtual void setup(Vertica::ServerInterface &srvInterface, const Vertica::SizedColumnType &argTypes) {
        ParamReader pSessionParams = srvInterface.getUDSessionParamReader("library");
        std::string rCount = pSessionParams.getStringRef("rowCount").str;
        rowCount = atoi(rCount.c_str());
    }
};
```
virtual void processBlock(Vertica::ServerInterface &srvInterface, Vertica::BlockReader &arg_reader, Vertica::BlockWriter &res_writer) {
    count = 0;
    if(arg_reader.getNumCols() != 2)
        vt_report_error(0, "Function only accepts two arguments, but %zu provided", arg_reader.getNumCols());
    do {
        const Vertica::vint a = arg_reader.getIntRef(0);
        const Vertica::vint b = arg_reader.getIntRef(1);
        res_writer.setInt(a+b);
        count++;
        res_writer.next();
    } while (arg_reader.next());
    srvInterface.log("count %d", count);
}

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes, SessionParamWriterMap &udParams) {
    rowCount = rowCount + count;
    std::ostringstream s;
    s << rowCount;
    const std::string i_as_string(s.str());
    udParams.getUDSessionParamWriter("library").getStringRef("rowCount").copy(i_as_string);
}

class RowCountsInfo : public Vertica::ScalarFunctionFactory {
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface) {
        return Vertica::vt_createFuncObject<RowCount>(srvInterface.allocator);
    }

    virtual void getPrototype(Vertica::ServerInterface &srvInterface, Vertica::ColumnTypes &argTypes, Vertica::ColumnTypes &returnType) {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }
};

RegisterFactory(RowCountsInfo);

Java Example: Defining Parameters

The following code fragment demonstrates adding a single parameter to the Java add2ints UDSF example. The getParameterType() method defines a single integer parameter that is
named constant.

```java
package com.mycompany.example;
import com.vertica.sdk.*;
public class Add2IntsWithConstantFactory extends ScalarFunctionFactory {
    @Override
    public void getPrototype(ServerInterface srvInterface,
                               ColumnTypes argTypes,
                               ColumnTypes returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }
    @Override
    public void getReturnType(ServerInterface srvInterface,
                               SizedColumnTypes argTypes,
                               SizedColumnTypes returnType)
    {
        returnType.addInt("sum");
    }
    // Defines the parameters for this UDSF. Works similarly to defining
    // arguments and return types.
    public void getParameterType(ServerInterface srvInterface,
                                  SizedColumnTypes parameterTypes)
    {
        // One INTEGER parameter named constant
        parameterTypes.addInt("constant");
    }
    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface)
    {
        return new Add2IntsWithConstant();
    }
}
```

See the Vertica Java SDK entry for `SizedColumnTypes` for a full list of the data-type-specific methods you can call to define parameters.

**Java Example: Using Session Parameters**

The RowCount example uses a user-defined session parameter, also called RowCount. This parameter counts the total number of rows processed by the UDx each time it runs. RowCount then displays the aggregate number of rows processed for all executions.

```java
package com.mycompany.example;
import com.vertica.sdk.*;
public class RowCountFactory extends ScalarFunctionFactory {
    //...
}
```
@Override
class RowCount extends ScalarFunction {
    private Integer count;
    private Integer rowCount;
    // In the setup method, you look for the rowCount parameter. If it doesn't exist, it is created.
    // Look in the default namespace which is "library," but it could be anything else, most likely
    "public" if not "library".
    @Override
    public void setup(ServerInterface srvInterface, SizedColumnTypes argTypes) {
        count = new Integer(0);
        ParamReader pSessionParams = srvInterface.getUDSessionParamReader("library");
        String rCount = pSessionParams.containsKey("rowCount")?
                pSessionParams.getString("rowCount"):
                "0";
        rowCount = Integer.parseInt(rCount);
    }
    @Override
    public void processBlock(ServerInterface srvInterface, BlockReader arg_reader, BlockWriter res_writer)
            throws UdfException, DestroyInvocation {
        do {
            ++count;
            long a = arg_reader.getLong(0);
            long b = arg_reader.getLong(1);
            res_writer.setLong(a+b);
            res_writer.next();
        } while (arg_reader.next());
    }
    @Override
    public void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes, SessionParamWriterMap udParams){
        rowCount = rowCount+count;
        udParams.getUDSessionParamWriter("library").setString("rowCount", Integer.toString(rowCount));
        srvInterface.log("RowNumber processed %d records", count);
    }
    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface){
        return new RowCount();
    }
}
Debugging Tips

You must thoroughly debug your UDx before deploying it to a production environment. The following tips can help you get your UDx ready for deployment.

Use a Single Node For Initial Debugging

You can attach to the Vertica process using a debugger such as gdb to debug your UDx code. Doing this in a multi-node environment, however, is very difficult. Therefore, consider setting up a single-node Vertica test environment to initially debug your UDx.

Write Messages to the Vertica Log

You can write to log files using the ServerInterface.log function. Every function in your UDx receives an instance of the ServerInterface object, so you can call the log function from anywhere in your UDx. The function acts similarly to printf, taking a formatted string, and an optional set of values and writing the string to a log file. Where the message is written depends on whether your function runs in fenced mode or unfenced mode:

- Functions running in unfenced mode write their messages into the vertica.log file in the catalog directory.
- Functions running in fenced mode write their messages into a log file named UDxLogs/UDxFencedProcesses.log in the catalog directory.

To help identify your function's output, Vertica adds the SQL function name bound to your UDx function to the log message.

The following code fragment shows how you can add a call to srvInterface.log in the Add2ints example code's processBlock() function to log its input values:

```c
const vint a = arg_reader.getIntRef(0);
const vint b = arg_reader.getIntRef(1);
srvInterface.log("got a: %d and b: %d", (int) a, (int) b);
```

This code generates an entries in the log file for each row the UDx processes. For example:

```
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 1 and b: 2
11-05-06 14:37:20.838 nameless:0x3f3a210 [UserMessage] <UDx> Add2ints - got a: 2 and b: 2
```
See Monitoring the Log Files in the Administrator's Guide for details on viewing the Vertica log files.

Developing with the C++ SDK

The Vertica SDK supports writing both fenced and unfenced UDxs of all types in C++ 11. You can download, compile, and run the examples; see Downloading and Running UDx Example Code. Running the examples is a good way to verify that your development environment has all needed libraries.

If you do not have access to a Vertica test environment, you can install Vertica on your development machine and run a single node. Each time you rebuild your UDx library, you need to re-install it into Vertica. The following diagram illustrates the typical development cycle.

Extending Vertica provides a high-level description of the key APIs in the SDK. For more details, see the C++ SDK Documentation.

Setting Up the C++ SDK

The Vertica C++ Software Development Kit (SDK) is distributed as part of the server installation. It contains the source and header files you need to create your UDx library. For examples that you can compile and run, see Downloading and Running UDx Example Code. For requirements for your development environment, see Setting Up a Development Environment.

The SDK files are located in the sdk subdirectory under the root Vertica server directory (usually, /opt/vertica/sdk). This directory contains a subdirectory, include, which contains the headers and source files needed to compile UDx libraries.

There are two files in the include directory you need when compiling your UDx:
Vertica.h is the main header file for the SDK. Your UDx code needs to include this file in order to find the SDK's definitions.

Vertica.cpp contains support code that needs to be compiled into the UDx library.

Much of the Vertica SDK API is defined in the VerticaUDx.h header file (which is included by the Vertica.h file). If you're curious, you might want to review the contents of this file in addition to reading the API documentation.

Finding the Current SDK Version

You must develop your UDx using the same SDK version as the database in which you plan to use it. To display the SDK version currently installed on your system, run the following command in vsql:

```sql
=> SELECT sdk_version();
```

Running the Examples

You can download the examples from the GitHub repository (see Downloading and Running UDx Example Code). Compiling and running the examples helps you to ensure that your development environment is properly set up.

To compile all of the examples, including the Java examples, issue the following command in the Java-and-C++ directory under the examples directory:

```
$ make
```

Note: To compile the examples, you must have a g++ development environment installed. To install a g++ development environment on Red Hat systems, run `yum install gcc gcc-c++ make`.

Compiling Your C++ Library

GNU g++ is the only supported compiler for compiling UDx libraries. Always compile your UDx code on the same version of Linux that you use on your Vertica cluster.

When compiling your library, you must always:
Pass the -shared and -fPIC flags to the linker. The simplest method is to just pass these flags to g++ when you compile and link your library.

Use the -Wno-unused-value flag to suppress warnings when macro arguments are not used. If you do not use this flag, you may get "left-hand operand of comma has no effect" warnings.

Compile sdk/include/Vertica.cpp and link it into your library. This file contains support routines that help your UDx communicate with Vertica. The easiest way to do this is to include it in the g++ command to compile your library. Vertica supplies this file as C++ source rather than a library to limit library compatibility issues.

Add the Vertica SDK include directory in the include search path using the g++ -I flag.

The SDK examples include a working makefile. See Downloading and Running UDx Example Code.

Example of Compiling a UDx

The following command compiles a UDx contained in a single source file named MyUDx.cpp into a shared library named MyUDx.so:

```
g++ -I/opt/vertica/sdk/include -Wall -shared -Wno-unused-value -fPIC -o MyUDx.so MyUDx.cpp /opt/vertica/sdk/include/Vertica.cpp
```

**Important:** Vertica only supports UDx development on 64-bit architectures.

After you debug your UDx, you are ready to deploy it. Recompile your UDx using the -O3 flag to enable compiler optimization.

You can add additional source files to your library by adding them to the command line. You can also compile them separately and then link them together.

**Tip:** The examples subdirectory in the Vertica SDK directory contains a make file that you can use as starting point for your own UDx project.

Handling External Libraries

You must link your UDx library to any supporting libraries that your UDx code relies on. These libraries might be either ones you developed or others provided by third parties. You have two options for linking:
• Statically link the support libraries into your UDx. The benefit of this method is that your UDx library does not rely on external files. Having a single UDx library file simplifies deployment because you just transfer a single file to your Vertica cluster. This method’s main drawback is that it increases the size of your UDx library file.

• Dynamically link the library to your UDx. You must sometimes use dynamic linking if a third-party library does not allow static linking. In this case, you must copy the libraries to your Vertica cluster in addition to your UDx library file.

Adding Metadata to C++ Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on a Vertica Analytic Database cluster and lets third-party users of your function know who created the function. Your library’s metadata appears in the USER_LIBRARIES system table after your library has been loaded into the Vertica Analytic Database catalog.

You declare the metadata for your library by calling the `RegisterLibrary()` function in one of the source files for your UDx. If there is more than one function call in the source files for your UDx, whichever gets interpreted last as Vertica Analytic Database loads the library is used to determine the library’s metadata.

The `RegisterLibrary()` function takes eight string parameters:

```c
RegisterLibrary(author,
    library_build_tag,
    library_version,
    library_sdk_version,
    source_url,
    description,
    licenses_required,
    signature);
```

• `author` contains whatever name you want associated with the creation of the library (your own name or your company's name for example).

• `library_build_tag` is a string you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when the library was compiled). This is useful for tracking instances of your library as you are developing them.

• `library_version` is the version of your library. You can use whatever numbering or naming scheme you want.
- **library_sdk_version** is the version of the Vertica Analytic Database SDK Library for which you've compiled the library.

  Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytic Database server. The version of the Vertica Analytic Database SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytic Database server uses to determine if your library is compatible with it.

- **source_url** is a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.

- **description** is a concise description of your library.

- **licenses_required** is a placeholder for licensing information. You must pass an empty string for this value.

- **signature** is a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

For example, the following code demonstrates adding metadata to the Add2Ints example (see **C++ Example: Add2Ints**).

```cpp
// Include the top-level Vertica SDK file
#include "Vertica.h"
// Using the Vertica namespace means we don't have to prefix all
// class references with Vertica::
using namespace Vertica;
/*
 * ScalarFunction implementation for a UDF that adds
 * two numbers together.
 */
class Add2Ints : public ScalarFunction
{
    public:
    /*
     * This function does all of the actual processing for the UDF.
     * In this case, it simply reads two integer values and returns
     * their sum.
     *
     * The inputs are retrieved via arg_reader
     * The outputs are returned via arg_writer
     */
    virtual void processBlock(ServerInterface &srvInterface,
        BlockReader &arg_reader,
        BlockWriter &res_writer)
    {
```
// While we have input to process
  do
  {
    // Read the two integer input parameters by calling the
    // BlockReader.getIntRef class function
    const vint a = arg_reader.getIntRef(0);
    const vint b = arg_reader.getIntRef(1);
    // Call BlockWriter.setInt to store the output value, which is the
    // two input values added together
    res_writer.setInt(a+b);
    // Finish writing the row, and advance to the next output row
    res_writer.next();
    // Continue looping until there are no more input rows
  }
  while (arg_reader.next());
}

/*
* This class provides metadata about the ScalarFunction class, and
* also instantiates a member of that class when needed.
*/
class Add2IntsFactory : public ScalarFunctionFactory
{
  // return an instance of Add2Ints to perform the actual addition.
  virtual ScalarFunction *createScalarFunction(ServerInterface &interface) {
    // Calls the vt_createFuncObj to create the new Add2Ints class instance.
    return vt_createFuncObj(interface allocator, Add2Ints);}

  // This function returns the description of the input and outputs of the
  // Add2Ints class's processBlock function. It stores this information in
  // two ColumnTypes objects, one for the input parameters, and one for
  // the return value.
  virtual void getPrototype(ServerInterface &interface,
                             ColumnTypes &argTypes,
                             ColumnTypes &returnType) {
    // Takes two ints as inputs, so add ints to the argTypes object
    argTypes.addInt();
    argTypes.addInt();
    // returns a single int, so add a single int to the returnType object.
    // Note that ScalarFunctions *always* return a single value.
    returnType.addInt();
  }
};

// Register the factory with Vertica
RegisterFactory(Add2IntsFactory);

// Register the library’s metadata.
RegisterLibrary("Whizzo Analytics Ltd.",
               "1234",
               "2.0",
               "7.0.0",
               "http://www.example.com/add2ints",
               "Add 2 Integer Library",
               ",",
               ",");
Loading the library and querying the USER_LIBRARIES system table shows the metadata supplied in the call to RegisterLibrary():

```sql
=> CREATE LIBRARY add2intslib AS '/home/dbadmin/add2ints.so';
CREATE LIBRARY
=> \x
Expanded display is on.
=> SELECT * FROM USER_LIBRARIES WHERE lib_name = 'add2intslib';
- [ RECORD 1 ]-+-----------------------------------------------------
schema_name | public
lib_name     | add2intslib
lib_oid      | 45035996273869808
author       | Whizzo Analytics Ltd.
owner_id     | 45035996273704962
lib_file_name| public_add2intslib_45035996273869808.so
md5_sum      | 732c9e145d447c8ac6e7304313d3b8a0
sdk_version  | v7.0.0-20131105
revision     | 125200
lib_build_tag| 1234
lib_version  | 2.0
lib_sdk_version | 7.0.0
source_url   | http://www.example.com/add2ints
description  | Add 2 Integer Library
licenses_required | signature |
```

**C++ SDK Data Types**

The Vertica SDK has typedefs and classes for representing Vertica data types within your UDx code. Using these typedefs ensures data type compatibility between the data your UDx processes and generates and the Vertica database. The following table describes some of the typedefs available. Consult the Vertica C++ SDK Documentation for a complete list, as well as lists of helper functions to convert and manipulate these data types.

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
<td>A Vertica interval</td>
</tr>
<tr>
<td>IntervalYM</td>
<td>A Vertica year-to-month interval.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>A Vertica timestamp</td>
</tr>
<tr>
<td>vint</td>
<td>A standard Vertica 64-bit integer</td>
</tr>
<tr>
<td>vint_null</td>
<td>A null value for integer values</td>
</tr>
<tr>
<td>vbool</td>
<td>A Boolean value in Vertica</td>
</tr>
<tr>
<td>Type Definition</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>vbool_null</td>
<td>A null value for a Boolean data types</td>
</tr>
<tr>
<td>vfloat</td>
<td>A Vertica floating point value</td>
</tr>
</tbody>
</table>
| VString         | String data types (such as varchar and char)  
|                 | **Note:** Do not use a VString object to hold an intermediate result. Use a std::string or char[] instead. |
| VNumeric        | Fixed-point data types from Vertica |

**Notes**

- When making some Vertica SDK API calls (such as VerticaType::getNumericLength()) on objects, make sure they have the correct data type. To minimize overhead and improve performance, most of the APIs do not check the data types of the objects on which they are called. Calling a function on an incorrect data type can result in an error.

- A NULL Vertica value string data type is converted into an empty C++ string.

- You cannot create instances of VString or VNumeric yourself. You can manipulate the values of existing objects of these classes that Vertica passes to your UDx, and extract values from them. However, only Vertica can instantiate these classes.

**Handling Errors**

If your UDx encounters some sort of error, it can report it back to Vertica using the vt_report_error macro. When called, this macro halts the execution of the UDx and causes the statement that called the function to fail. The macro takes two parameters: an error number and an error message string. Both the error number and message appear in the error that Vertica reports to the user. The error number is not defined by Vertica. You can use whatever value that you wish.

For example, the following ScalarFunction class divides two integers. To prevent division by zero, it tests the second parameter. If it is zero, the function reports the error back to Vertica.

```c++
/*  
 * Demonstrate reporting an error  
 */
```
class Div2ints : public ScalarFunction
{
public:
  virtual void processBlock(ServerInterface &srvInterface,
                             BlockReader &arg_reader,
                             BlockWriter &res_writer)
  {
    // While we have inputs to process
    do
    {
      const vint a = arg_reader.getIntRef(0);
      const vint b = arg_reader.getIntRef(1);
      if (b == 0)
      {
        vt_report_error(1, "Attempted divide by zero");
      }
      res_writer.setInt(a/b);
      res_writer.next();
    } while (arg_reader.next());
  }
};

Loading and invoking the function demonstrates how the error appears to the user.

=> CREATE LIBRARY Div2IntsLib AS '/home/dbadmin/Div2ints.so';
CREATE LIBRARY
=> CREATE FUNCTION div2ints AS LANGUAGE 'C++' NAME 'Div2intsInfo' LIBRARY Div2IntsLib;
CREATE FUNCTION
=> SELECT div2ints(25, 5);
 div2ints
   --------
     5
 (1 row)
=> SELECT * FROM MyTable;
 a | b
  | |
---+---
12 | 6
 7 | 0
12 | 2
18 | 9
 (4 rows)
=> SELECT * FROM MyTable WHERE div2ints(a, b) > 2;
ERROR: Error in calling processBlock() for User Defined Scalar Function div2ints at Div2ints.cpp:21, error code: 1, message: Attempted divide by zero

Your function must not allow an exception to be passed back to Vertica. You should use a top-level try-catch block to catch any stray exceptions that might be thrown by your code or any functions or libraries your code calls. This is especially important when running a UDx in unfenced mode. Any errors in an unfenced UDx can result in database instability or even data loss.
Handling Cancel Requests

You can cancel a query that calls your UDx (usually, by by pressing CTRL+C in vsqI). How Vertica handles the cancelation of the query and your UDx depends on whether your UDx is running in fenced or unfenced mode:

- If your UDx is running in unfenced mode, Vertica either stops the function when it requests a new block of input or output, or waits until your function completes running and discards the results.

- If your UDx is running in Fenced Mode, Vertica kills the zygote process that is running your function if it continues processing past a timeout.

See Fenced Mode for more information about running functions in fenced mode.

To give you more control over what happens to your function when the user cancels its query, the Vertica SDK includes an API for some UDxs to handle cancelation. Any function class that inherits from the Vertica::UDXObjectCancelable class can test whether the query calling it has been canceled using a function named isCanceled(). Your function can also implement a callback function named cancel() that Vertica calls when the function's query is canceled. Currently, the two classes that inherit from UDXObjectCancelable are TransformFunction and AnalyticFunction.

Exiting When the Calling Query Has Been Canceled

The processPartition() function in your User-Defined Transform Function (UDTF) or Analytic Function (UDAnF) can call Vertica::UDXObjectCancelable.isCanceled() to determine if the user has canceled the query that called it. If isCanceled() returns true, the query has been canceled and your processPartition() function should exit immediately to prevent it from wasting CPU time. If your UDx is not running in Fenced Mode, Vertica cannot halt your function, and has to wait for it to finish. If it is running in fenced mode, Vertica can eventually kill the side process running it, but not until it has wasted some processing time.

How often your processPartition() function calls isCanceled() depends on how much processing it performs on each row of data. Calling isCanceled() does add some overhead to your function, so you shouldn't call it too often. For transforms that do not perform lengthy processing, you could check for cancellation every 100 or 1000 rows or so. If your processPartition() function performs extensive processing for each row, you may want to check isCanceled() every 10 or so rows.
The following code fragment shows how you could have the StringTokenizer UDTF example check whether its query has been canceled:

```java
// The primary class for the StringTokenizer UDTF.
class StringTokenizer : public TransformFunction {
    // Called for each partition in the table. Recieves the data from
    // The source table and
    virtual void processPartition(ServerInterface &srvInterface,
                                   PartitionReader &inputReader,
                                   PartitionWriter &outputWriter) {
        try {
            // Loop through the input rows
            int rowCount = 0; // Count the number of rows processed.
            do {
                rowCount++; // Processing a new row of data
                // Check for cancelation every 100 rows.
                if (rowCount % 100 == 0) {
                    if (isCanceled()) // See if query has been canceled
                    {
                        // Log cancelation
                        srvInterface.log("Got canceled!");
                        return; // Exit out of UDTF immediately.
                    }
                }
            } // Rest of the function here
            . . .
        }
    }
}
```

This example checks for cancellation after processing 100 rows in the partition of data. If the query has been canceled, the example logs a message, then returns to the caller to exit the function.

**Note:** You need to strike a balance between adding overhead to your functions by calling `isCanceled()` and having your functions waste CPU time by running after their query has been canceled (usually, a rare event). For functions such as `StringTokenizer` which have a low overall processing cost, it usually does not make sense to test for cancellation. The cost of adding overhead to all function calls outweighs the amount of resources wasted by having the function run to completion or having its zygote process killed by Vertica on the rare occasions that its query is canceled.

### Implementing the Cancel Callback Function

Your User-Defined Transform Function (UDTF) or Analytic Function (UDAnF) can implement a `cancel()` callback function that Vertica calls if the query that called the function has been canceled. You usually implement this function to perform an orderly shutdown of any additional processing that your UDx spawned. For example, you can have your `cancel()` function shut down threads that your UDx has spawned or signal a third-party library that it needs to stop processing and exit. Your `cancel()` function should leave your UDx's function
class ready to be destroyed, since Vertica calls the UDx's destroy function after the cancel function has exited.

Notes

- If your UDTF or UDAnF does not implement cancel(), Vertica assumes your UDx does not need to perform any special cancel processing, and calls the function class's destroy() function to have it free any resources (see Resource Use for C++ UDxs).

- Your cancel() function is called from a different thread than the thread running your UDx's processPartition() function.

- The call to the cancel() function is not synchronized in any way with your UDx's processPartition() function. If you need your processPartition() function to exit before your cancel() function performs some action (killing threads, for example) you need to have the two function synchronize their actions.

- If your cancel() function runs for too long, Vertica kills the side process running your function, if it is running in Fenced Mode.
Resource Use for C++ UDxs

Your UDxs consume at least a small amount of memory by instantiating classes and creating local variables. This basic memory usage by UDxs is small enough that you do not need to be concerned about it.

If your UDx needs to allocate more than one or two megabytes of memory for data structures, or requires access to additional resources such as files, you must inform Vertica about its resource use. Vertica can then ensure that the resources your UDx requires are available before running a query that uses it. Even moderate memory use (10MB per invocation of a UDx, for example) can become an issue if there are many simultaneous queries that call it.

Note: If your UDx allocates its own memory, you must make absolutely sure it properly frees it. Failing to free even a single byte of allocated memory can have huge consequences if your UDx is called to operate on a multi-million-row table. Instead of having your code allocate its own memory, you should use the C++ vt_alloc macro, which uses Vertica's own memory manager to allocate and track memory. This memory is guaranteed to be properly disposed of when your UDx completes execution. See Allocating Resources for UDxs for more information.

Allocating Resources for UDxs

You have two options for allocating memory and file handles for your User-Defined Extensions (UDxs):

- Use Vertica SDK macros to allocate resources. This is the best method, since it uses Vertica's own resource manager, and guarantees that resources used by your UDx are reclaimed. See Allocating Resources with the SDK Macros.

- Allocate resources in your UDxs yourself using standard C++ methods (instantiating objects using new, allocating memory blocks using malloc(), etc.). You must manually free these resources before your UDx exits.

Note: You must be extremely careful if you choose to allocate your own resources in your UDx. Failing to free resources properly will have significant negative impact, especially if your UDx is running in unfenced mode.

Whichever method you choose, you usually allocate resources in a function named setup() in your UDx class. This function is called after your UDx function object is instantiated, but before Vertica calls it to process data.
If you allocate memory on your own in the `setup()` function, you must free it in a corresponding function named `destroy()`. This function is called after your U Dx has performed all of its processing. This function is also called if your U Dx returns an error (see Handling Errors).

**Note:** Always use the `setup()` and `destroy()` functions to allocate and free resources instead of your own constructors and destructors. The memory for your U Dx object is allocated from one of Vertica's own memory pools. Vertica always calls your U Dx's `destroy()` function before it deallocates the object's memory. There is no guarantee that your U Dx's destructor is will be called before the object is deallocated. Using the `destroy()` function ensures that your U Dx has a chance to free its allocated resources before it is destroyed.

The following code fragment demonstrates allocating and freeing memory using a `setup()` and `destroy()` function.

```cpp
class MemoryAllocationExample : public ScalarFunction
{
public:
    uint64* myarray;
    // Called before running the UDF to allocate memory used throughout
    // the entire UDF processing.
    virtual void setup(ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
    {
        try
        {
            // Allocate an array. This memory is directly allocated, rather than
            // letting Vertica do it. Remember to properly calculate the amount
            // of memory you need based on the data type you are allocating.
            // This example divides 500MB by 8, since that's the number of
            // bytes in a 64-bit unsigned integer.
            myarray = new uint64[1024 * 1024 * 500 / 8];
        }
        catch (std::bad_alloc &ba)
        {
            // Always check for exceptions caused by failed memory
            // allocations.
            vt_report_error(1, "Couldn't allocate memory :[%s]", ba.what());
        }
    }

    // Called after the UDF has processed all of its information. Use to free
    // any allocated resources.
    virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes)
    {
        try
        {
            // Properly dispose of the allocated memory.
            delete[] myarray;
        }
    }
};
```
catch (std::bad_alloc &ba)
{
    // Always check for exceptions caused by failed memory
    // allocations.
    vt_report_error(1, "Couldn't free memory :[%s]", ba.what());
}

Allocating Resources with the SDK Macros

The Vertica SDK provides three macros to allocate memory:

- **vt_alloc** allocates a block of memory to fit a specific data type (vint, struct, etc.).
- **vt.alloc** allocates a block of memory to hold an array of a specific data type.
- **vt.alloc** allocates an arbitrarily-sized block of memory.

All of these macros allocate their memory from memory pools managed by Vertica. The main benefit of allowing Vertica to manage your UDx's memory is that the memory is automatically reclaimed after your UDx has finished. This ensures there is no memory leaks in your UDx.

Because Vertica frees this memory automatically, do not attempt to free any of the memory you allocate through any of these macros. Attempting to free this memory results in run-time errors.

Informing Vertica of Resource Requirements

When you run your UDx in fenced mode, Vertica monitors its use of memory and file handles. If your UDx uses more than a few megabytes of memory or any file handles, it should tell Vertica about its resource requirements. Knowing the resource requirements of your UDx allows Vertica to determine whether it can run the UDx immediately or needs to queue the request until enough resources become available to run it.

Determining how much memory your UDx requires can be difficult in some cases. For example, if your UDx extracts unique data elements from a data set, there is potentially no bound on the number of data items. In this case, a useful technique is to run your UDx in a test environment and monitor its memory use on a node as it handles several differently-sized queries, then extrapolate its memory use based on the worst-case scenario it may face in your production environment. In all cases, it's usually a good idea to add a safety margin to the amount of memory you tell Vertica your UDx uses.
Note: The information on your UDx's resource needs that you pass to Vertica is used when planning the query execution. There is no way to change the amount of resources your UDx requests from Vertica while the UDx is actually running.

Your UDx informs Vertica of its resource needs by implementing the getPerInstanceResources() function in its factory class (see Vertica::UDXFactory::getPerInstanceResources() in the SDK documentation). If your UDx's factory class implements this function, Vertica calls it to determine the resources your UDx requires.

The getPerInstanceResources() function receives an instance of the Vertica::VResources struct. This struct contains fields that set the amount of memory and the number of file handles your UDx needs. By default, the Vertica server allocates zero bytes of memory and 100 file handles for each instance of your UDx.

Your implementation of the getPerInstanceResources() function sets the fields in the VResources struct based on the maximum resources your UDx may consume for each instance of the UDx function. So, if your UDx's processBlock() function creates a data structure that uses at most 100MB of memory, your UDx must set the VResources.scratchMemory field to at least 104857600 (the number of bytes in 100MB). Leave yourself a safety margin by increasing the number beyond what your UDx should normally consume. In this example, allocating 115000000 bytes (just under 110MB) is a good idea.

The following ScalarFunctionFactory class demonstrates calling getPerInstanceResources() to inform Vertica about the memory requirements of the MemoryAllocationExample class shown in Allocating Resources for UDxs. It tells Vertica that the UDSF requires 510MB of memory (which is a bit more than the UDSF actually allocates, to be on the safe size).

class MemoryAllocationExampleFactory : public ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    {
        return vt_createFuncObj(srvInterface.allocator, MemoryAllocationExample);
    }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
                                Vertica::ColumnTypes &argTypes,
                                Vertica::ColumnTypes &returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    // Tells Vertica the amount of resources that this UDF uses.
    virtual void getPerInstanceResources(ServerInterface &srvInterface,
                                           VResources &res)
    {

res.scratchMemory += 1024LL * 1024 * 510; // request 510MB of memory
}

Setting Memory Limits for Fenced-Mode UDxs

Vertica calls a fenced-mode UDx's implementation of Vertica::UDXFactory::getPerInstanceResources() to determine if there are enough free resources to run the query containing the UDx (see Informing Vertica of Resource Requirements). Since these reports are not generated by actual memory use, they can be inaccurate. Once started by Vertica, a UDx could allocate far more memory or file handles than it reported it needs.

The FencedUDxMemoryLimitMB configuration parameter lets you create an absolute memory limit for UDxs. Any attempt by a UDx to allocate more memory than this limit results in a bad_alloc exception. For more information on configuration parameters, see Configuration Parameters in the Administrator's Guide. For an example of setting FencedUDxMemoryLimitMB, see How Resource Limits Are Enforced.

How Resource Limits Are Enforced

Before running a query, Vertica determines how much memory it requires to run. If the query contains a fenced-mode UDx which implements the getPerInstanceResources() function in its factory class, Vertica calls it to determine the amount of memory the UDx needs and adds this to the total required for the query. Based on these requirements, Vertica decides how to handle the query:

- If the total amount of memory required (including the amount that the UDxs report that they need) is larger than the session's MEMORYCAP or resource pool's MAXMEMORYSIZE setting, Vertica rejects the query. For more information about resource pools, see Resource Pool Architecture in the Administrator's Guide.

- If the amount of memory is below the limit set by the session and resource pool limits, but there is currently not enough free memory to run the query, Vertica queues it until enough resources become available.

- If there are enough free resources to run the query, Vertica executes it.

Note: Vertica has no other way to determine the amount of resources a UDx requires other than the values it reports using the getPerInstanceResources() function. A
UDX could use more resources than it claims, which could cause performance issues for other queries that are denied resources. You can set an absolute limit on the amount of memory UDXs can allocate. See Setting Memory Limits for Fenced-Mode UDXs for more information.

If the process executing your UDX attempts to allocate more memory than the limit set by the FencedUDxMemoryLimitMB configuration parameter, it receives a bad_alloc exception. For more information about FencedUDxMemoryLimitMB, see Setting Memory Limits for Fenced-Mode UDXs.

Below is the output of loading a UDSF that consumes 500MB of memory, then changing the memory settings to cause out-of-memory errors. The MemoryAllocationExample UDSF in the following example is just the Add2Ints UDSF example altered as shown in Allocating Resources for UDXs and Informing Vertica of Resource Requirements to allocate 500MB of RAM.

```sql
=> CREATE LIBRARY mylib AS '/home/dbadmin/MemoryAllocationExample.so';
CREATE LIBRARY
=> CREATE FUNCTION usemem AS NAME 'MemoryAllocationExampleFactory' LIBRARY mylib
  -> FENCED;
CREATE FUNCTION
=> SELECT usemem(1,2);
usemem
--------
3
(1 row)
```

The following statements demonstrate setting the session's MEMORYCAP to lower than the amount of memory that the UDSF reports it uses. This causes Vertica to return an error before it executes the UDSF.

```sql
=> SET SESSION MEMORYCAP '100M';
SET
=> SELECT usemem(1,2);
ERROR 3596: Insufficient resources to execute plan on pool sysquery
[Request exceeds session memory cap: 520328KB > 102400KB]
=> SET SESSION MEMORYCAP = default;
SET
```

The resource pool can also prevent a UDX from running if it requires more memory than is available in the pool. The following statements demonstrate the effect of creating and using a resource pool that has too little memory for the UDSF to run. Similar to the session's MAXMEMORYCAP limit, the pool's MAXMEMORYSIZE setting prevents Vertica from executing the query containing the UDSF.

```sql
=> CREATE RESOURCE POOL small MEMORYSIZE '100M' MAXMEMORYSIZE '100M';
CREATE RESOURCE POOL
=> SET SESSION RESOURCE POOL small;
SET
=> CREATE TABLE ExampleTable(a int, b int);
```
CREATE TABLE
=> INSERT /*+direct*/ INTO ExampleTable VALUES (1,2);
OUTPUT
--------
1
(1 row)
=> SELECT usemem(a, b) FROM ExampleTable;
ERROR 3596: Insufficient resources to execute plan on pool small 
[Request Too Large:Memory(KB) Exceeded: Requested = 523136, Free = 102400 (Limit = 102400, Used = 0)]
=> DROP RESOURCE POOL small; --Dropping the pool resets the session's pool
DROP RESOURCE POOL

Finally, setting the FencedUDxMemoryLimitMB configuration parameter to lower than the UDx actually allocates results in the UDx throwing an exception. This is a different case than either of the previous two examples, since the query actually executes. The UDx's code needs to catch and handle the exception. In this example, it uses the vt_report_error macro to report the error back to Vertica and exit.

=> ALTER DATABASE mydb SET FencedUDxMemoryLimitMB = 300;
=> SELECT usemem(1,2);
    ERROR 3412: Failure in UDx RPC call InvokeSetup(): Error calling setup() in User Defined Object [usemem] at [MemoryAllocationExample.cpp:32], error code: 1, message: Couldn't allocate memory :[std::bad_alloc]
=> ALTER DATABASE mydb SET FencedUDxMemoryLimitMB = -1;
=> SELECT usemem(1,2);
    usemem
--------
    3
(1 row)

See Also

- SET SESSION RESOURCE_POOL
- SET SESSION MEMORYCAP
- SET_CONFIG_PARAMETER

Developing with the Java SDK

The Vertica SDK supports writing Java UDxs of all types except aggregate functions. All Java UDxs are fenced.

You can download, compile, and run the examples; see Downloading and Running UDx Example Code. Running the examples is a good way to verify that your development environment has all needed libraries.
If you do not have access to a Vertica test environment, you can install Vertica on your development machine and run a single node. Each time you rebuild your UDx library, you need to re-install it into Vertica. The following diagram illustrates the typical development cycle.

Extending Vertica provides a high-level description of the key APIs in the SDK. For more details, see the Java SDK Documentation.

**Setting Up the Java SDK**

The Vertica Java Software Development Kit (SDK) is distributed as part of the server installation. It contains the source and JAR files you need to create your UDx library. For examples that you can compile and run, see Downloading and Running UDx Example Code. For requirements for your development environment, see Setting Up a Development Environment.

To use the SDK you need two files from the Java support package:

- `/opt/vertica/bin/VerticaSDK.jar` contains the Vertica Java SDK and other supporting files.
- `/opt/vertica/sdk/BuildInfo.java` contains version information about the SDK. You must compile this file and include it within your Java UDx JAR files.

If you are not doing your development on a database node, you can copy these two files from one of the database nodes to your development system.

The `BuildInfo.java` and `VerticaSDK.jar` files that you use to compile your UDx must be from the same SDK version. Both files must also match the version of the SDK files on your Vertica hosts. Versioning is only an issue if you are not compiling your UDxs on a Vertica host. If you are compiling on a separate development system, always refresh your copies of these two files and recompile your UDxs just before deploying them.

**Finding the Current SDK Version**

You must develop your UDx using the same SDK version as the database in which you plan to use it. To display the SDK version currently installed on your system, run the following command in vsql:
SELECT sdk_version();

Compiling BuildInfo.java

You need to compile the `BuildInfo.java` file into a class file, so you can include it in your Java UDx JAR library. If you are using a Vertica node as a development system, you can either:

- Copy the `BuildInfo.java` file to another location on your host.
- If you have root privileges, compile the `BuildInfo.java` file in place. (Only the root user has privileges to write files to the `/opt/vertica/sdk` directory.)

Compile the file using the following command. Replace `path` with the path to the file and `output-directory` with the directory where you will compile your UDxs.

```
$ javac -classpath /opt/vertica/bin/VerticaSDK.jar /path/BuildInfo.java -d output-directory
```

If you use an IDE such as Eclipse, you can include the `BuildInfo.java` file in your project instead of compiling it separately. You must also add the `VerticaSDK.jar` file to the project's build path. See your IDE's documentation for details on how to include files and libraries in your projects.

Running the Examples

You can download the examples from the GitHub repository (see Downloading and Running UDx Example Code). Compiling and running the examples helps you to ensure that your development environment is properly set up.

If you have not already done so, set the JAVA_HOME environment variable to your JDK (not JRE) directory.

To compile all of the examples, including the Java examples, issue the following command in the `Java-and-C++` directory under the examples directory:

```
$ make
```

To compile only the Java examples, issue the following command in the `Java-and-C++` directory under the examples directory:

```
$ make JavaFunctions
```
To compile the examples, you must have make installed. To install make on Red Hat systems, run yum install make.

Compiling and Packaging a Java Library

Before you can use your Java UDx, you need to compile it and package it into a JAR file. The SDK examples include a working makefile. See Downloading and Running UDx Example Code.

Compile Your Java UDx

You must include the SDK JAR file in the classpath when you compile your Java UDx source files so the Java compiler can resolve the Vertica API calls. If you are using the command-line Java compiler on a host in your database cluster, enter this command:

```
$ javac -classpath /opt/vertica/bin/VerticaSDK.jar factorySource.java 
[functionSource.java...] -d output-directory
```

If all of your source files are in the same directory, you can use *.java on the command line instead of listing the files individually.

If you are using an IDE, verify that a copy of the VerticaSDK.jar file is in the build path.

UDx Class File Organization

After you compile your UDx, you must package its class files and the BuildInfo.class file into a JAR file.

Note: You can package as many UDxs as you want into the same JAR file. Bundling your UDxs together saves you from having to load multiple libraries.

To use the jar command packaged as part of the JDK, you must organize your UDx class files into a directory structure matching your class package structure. For example, suppose your UDx's factory class has a fully-qualified name of com.mycompany.udfs.Add2ints. In this case, your class files must be in the directory hierarchy com/mycompany/udfs relative to your project's base directory. In addition, you must have a copy of the BuildInfo.class file in the path com/vertica/sdk so that it can be included in the JAR file. This class must appear in your JAR file to indicate the SDK version that was used to compile your Java UDx.
The JAR file for the Add2ints UDSF example explained in Java Example: Add2Ints has the following directory structure after compilation:

- com/vertica/sdk/BuildInfo.class
- com/mycompany/example/Add2intsFactory.class
- com/mycompany/example/Add2intsFactory$Add2 ints.class

Package Your UDx Into a JAR File

To create a JAR file from the command line:

1. Change to the root directory of your project.
2. Use the jar command to package the BuildInfo.class file and all of the classes in your UDx:

```
# jar -cvf libname.jar com/vertica/sdk/BuildInfo.class \
    packagePath/*.class
```

When you type this command, `libname` is the filename you have chosen for your JAR file (choose whatever name you like), and `packagePath` is the path to the directory containing your UDx's class files.

- For example, to package the files from the Add2ints example, you use the command:

```
# jar -cvf Add2intsLib.jar com/vertica/sdk/BuildInfo.class \
    com/mycompany/example/*.class
```

- More simply, if you compiled `BuildInfo.class` and your class files into the same root directory, you can use the following command:

```
# jar -cvf Add2intsLib.jar .
```

You must include all of the class files that make up your UDx in your JAR file. Your UDx always consists of at least two classes (the factory class and the function class). Even if you defined your function class as an inner class of your factory class, Java generates a separate class file for the inner class.

After you package your UDx into a JAR file, you are ready to deploy it to your Vertica database.
Handling Java UDx Dependencies

If your Java UDx relies on one or more external libraries, you can handle the dependencies in one of three ways:

- Bundle the JAR files into your UDx JAR file using a tool such as One-JAR or Eclipse Runnable JAR Export Wizard.

- Unpack the JAR file and then repack its contents in your UDx's JAR file.

- Copy the libraries to your Vertica cluster in addition to your UDx library. Then, use the DEPENDS keyword of the CREATE LIBRARY statement to tell Vertica that the UDx library depends on the external libraries. This keyword acts as a library-specific CLASSPATH setting. Vertica distributes the support libraries to all of the nodes in the cluster and sets the class path for the UDx so it can find them.

If your UDx depends on native libraries (SO files), use the DEPENDS keyword to specify their path. When you call System.loadLibrary in your UDx (which you must do before using a native library), this function uses the DEPENDS path to find them. You do not need to also set the LD_LIBRARY_PATH environment variable.

Important: In previous versions of Vertica, the JavaClassPathForUDx configuration parameter set the locations of external libraries for all Java UDxs. In Vertica Version 7.1, this configuration parameter is deprecated. It will be removed in a future version. If you upgrade from a previous version of Vertica, switch any existing Java UDxs that rely on support libraries to use the new DEPENDS keyword. To make this switch, use the ALTER LIBRARY statement to recreate the UDx library. In this statement, supply the DEPENDS keyword to specify the directory containing the support libraries.

External Library Example

The following example demonstrates using an external library with a Java UDx.

The following sample code defines a simple class, named VowelRemover. It contains a single method, named removevowels, that removes all of the vowels (the letters a, e, i, o u, and y) from a string.

```java
package com.mycompany.libs;

public class VowelRemover {
```
public String removevowels(String input) {
    return input.replaceAll("(?i)[aeiouy]", ");
}

You can compile this class and package it into a JAR file with the following commands:

$ javac -g com/mycompany/libs/VowelRemover.java
$ jar cf mycompanylibs.jar com/mycompany/libs/VowelRemover.class

The following code defines a Java UDSF, named DeleteVowels, that uses the library defined in the preceding example code. DeleteVowels accepts a single VARCHAR as input, and returns a VARCHAR.

```java
package com.mycompany.udx;
// Import the support class created earlier
import com.mycompany.libs.VowelRemover;
// Import the Vertica SDK
import com.vertica.sdk.*;

public class DeleteVowelsFactory extends ScalarFunctionFactory {
    @Override
    public ScalarFunction createScalarFunction(ServerInterface arg0) {
        return new DeleteVowels();
    }

    @Override
    public void getPrototype(ServerInterface arg0, ColumnTypes argTypes,
        ColumnTypes returnTypes) {
        // Accept a single string and return a single string.
        argTypes.addVarchar();
        returnTypes.addVarchar();
    }

    @Override
    public void getReturnType(ServerInterface srvInterface,
        SizedColumnTypes argTypes,
        SizedColumnTypes returnType){
        returnType.addVarchar(
            // Output will be no larger than the input.
            argTypes.getColumnType(0).getStringLength(), "RemovedVowels");
    }

    public class DeleteVowels extends ScalarFunction {
        @Override
        public void processBlock(ServerInterface arg0, BlockReader argReader,
            BlockWriter resWriter) throws UdfException, DestroyInvocation {
            // Create an instance of the VowelRemover object defined in
            // the library.
            VowelRemover remover = new VowelRemover();
            do {
                String instr = argReader.getString(0);
                instr = remover.removevowels(instr);
                resWriter.getString(0, instr);
            } while (!argReader.isEndBlock());
        }
    }
}
```
// Call the removevowels method defined in the library.
resWriter.setString(remover.removevowels(instr));
resWriter.next();
} while (argReader.next());
}
}

Use the following commands to build the example UDSF and package it into a JAR:

- The first javac command compiles the SDK's BuildInfo class. Vertica requires all UDx libraries to contain this class. The javac command's -d option outputs the class file in the directory structure of your UDSF's source.
- The second javac command compiles the UDSF class. It adds the previously-created mycompanylibs.jar file to the class path so compiler can find the VowelRemover class.
- The jar command packages the BuildInfo and the classes for the UDx library together.

```
$ javac -g -cp /opt/vertica/bin/VerticaSDK.jar
   /opt/vertica/sdk/com/vertica/sdk/BuildInfo.java -d .
$ javac -g -cp mycompanylibs.jar:/opt/vertica/bin/VerticaSDK.jar
   com/mycompany/udx/DeleteVowelsFactory.java
$ jar cf DeleteVowelsLib.jar com/mycompany/udx/*.class
   com/vertica/sdk/*.class
```

To install the UDx library, you must copy both of the JAR files to a node in the Vertica cluster. Then, connect to the node to execute the CREATE LIBRARY statement.

The following example demonstrates how to load the UDx library after you copy the JAR files to the home directory of the dbadmin user. The DEPENDS keyword tells Vertica that the UDx library depends on the mycompanylibs.jar file.

```
=> CREATE LIBRARY DeleteVowelsLib AS
   '/home/dbadmin/DeleteVowelsLib.jar' DEPENDS '/home/dbadmin/mycompanylibs.jar'
   LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE FUNCTION deleteVowels AS language 'java' NAME
   'com.mycompany.udx.DeleteVowelsFactory' LIBRARY DeleteVowelsLib;
CREATE FUNCTION
=> SELECT deleteVowels('I hate vowels!');
  deleteVowels
--------------
  ht vwls!
(1 row)
```
Java and Vertica Data Types

The Vertica Java SDK converts Vertica's native data types into the appropriate Java data type. The following table lists the Vertica data types and their corresponding Java data types.

<table>
<thead>
<tr>
<th>Vertica Data Type</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>long</td>
</tr>
<tr>
<td>FLOAT</td>
<td>double</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>com.vertica.sdk.VNumeric</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>CHAR, VARCHAR, LONG VARCHAR</td>
<td>com.vertica.sdk.VString</td>
</tr>
<tr>
<td>BINARY, VARBINARY, LONG VARBINARY</td>
<td>com.vertica.sdk.VString</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.Timestamp</td>
</tr>
</tbody>
</table>

Note: Some Vertica data types are not supported.

Setting BINARY, VARBINARY, and LONG VARBINARY Values

The Vertica BINARY, VARBINARY, and LONG VARBINARY data types are converted as the Java UDx SDK's VString class. You can also set the value of a column with one of these data types with a ByteBuffer object (or a byte array wrapped in a ByteBuffer) using the PartitionWriter.setStringBytes() method. See the Java API UDx entry for PartitionWriter.setStringBytes() for more information.

Timestamps and Time Zones

When the SDK converts a Vertica timestamp into a Java timestamp, it uses the time zone of the JVM. If the JVM is running in a different time zone than the one used by Vertica, the results can be confusing.

Vertica stores timestamps in the database in UTC. (If a database time zone is set, the conversion is done at query time.) To prevent errors from the JVM time zone, add the following code to the processing method of your UDx:
Strings

The Java SDK contains a class named `StringUtils` that assists you when manipulating string data. One of its more useful features is its `getStringBytes()` method. This method extracts bytes from a `String` in a way that prevents the creation of invalid strings. If you attempt to extract a substring that would split part of a multi-byte UTF-8 character, `getStringBytes()` truncates it to the nearest whole character.

Handling NULL Values

Your UDxs must be prepared to handle NULL values. These values usually must be handled separately from regular values.

Reading NULL Values

Your UDX reads data from instances of the the `BlockReader` or `PartitionReader` classes. If the value of a column is NULL, the methods you use to get data (such as `getLong`) return a Java `null` reference. If you attempt to use the value without checking for NULL, the Java runtime will throw a null pointer exception.

You can test for null values before reading columns by using the data-type-specific methods (such as `isLongNull`, `isDoubleNull`, and `isBooleanNull`). For example, to test whether the `INTEGER` first column of your UDX's input is a NULL, you would use the statement:

```java
// See if the Long value in column 0 is a NULL
if (inputReader.isLongNull(0)) {
    // value is null
    ...
}
```

Writing NULL Values

You output NULL values using type-specific methods on the `BlockWriter` and `PartitionWriter` classes (such as `setLongNull` and `setStringNull`). These methods take the column number to receive the NULL value. In addition, the `PartitionWriter` class has data-type specific set value methods (such as `setLongValue` and `setStringValue`). If you pass these methods a value, they set the output column to that value. If you pass them a Java `null` reference, they set the output column to NULL.
Handling Errors

If your UDX encounters an unrecoverable error, it should instantiate and throw a UdfException. The exception causes the transaction containing the function call to be rolled back.

The UdfException constructor takes a numeric code (which can be anything you want since it is just reported in the error message) and an error message string. If you want to report additional diagnostic information about the error, you can write messages to a log file before throwing the exception (see Writing Messages to the Log File).

The following code fragment demonstrates adding error checking to the Add2ints UDSF example (shown in Java Example: Add2Ints). If either of the arguments is NULL, the processBlock() method throws an exception.

```java
@Override
public void processBlock(ServerInterface srvInterface,
    BlockReader argReader,
    BlockWriter resWriter)
    throws UdfException, DestroyInvocation
{
    do {
        // Test for NULL value. Throw exception if one occurs.
        if (argReader.isLongNull(0) || argReader.isLongNull(1)) {
            // No nulls allowed. Throw exception
            throw new UdfException(1234, "Cannot add a NULL value");
        }
    }
}
```

Note: This example isn't realistic, since you would likely just replace the NULL value with a zero or return a NULL value. Your UDX should only throw an exception if there is no way to compensate for the error.

When your UDX throws an exception, the side process running your UDX reports the error back to Vertica and exits. Vertica displays the error message contained in the exception and a stack trace to the user:

```sql
=> SELECT add2ints(2, NULL);
ERROR 3399: Failure in UDX RPC call InvokeProcessBlock(): Error in User Defined Object [add2ints], error code: 1234
com.vertica.sdk.UdfException: Cannot add a NULL value
    at com.mycompany.example.Add2intsFactory$Add2ints.processBlock(Add2intsFactory.java:37)
    at com.vertica.udxfence.UDxExecContext.processBlock(UDxExecContext.java:780)
    at com.vertica.udxfence.UDxExecContext.run(UDxExecContext.java:173)
    at java.lang.Thread.run(Thread.java:662)
```
Writing Messages to the Log File

Writing messages to a log is useful when you are debugging your Java UDxs, or you want to output additional information about an error condition. You can write messages to a log file by calling the `ServerInterface.log()` method, passing it a `printf()`-style String value along with any variables referenced in the string. (See the `java.util.Formatter` class documentation for details of formatting this string value.) An instance of the `ServerInterface` class is passed to the main processing method of every SDK class you can override.

The following code fragment demonstrates how you could log the values passed into the `Add2ints` UDFS example.

```java
@override
public void processBlock(ServerInterface srvInterface,
    BlockReader argReader,
    Blockwriter resWriter)
    throws UdfException, DestroyInvocation
{
    do {
        // Get the two integer arguments from the BlockReader
        long a = argReader.getLong(0);
        long b = argReader.getLong(1);
        // Log the input values
        srvInterface.log("Got values a=%d and b=%d", a, b);
    }
```

The messages are written to a log file stored in the catalog directory's `UDxLog` subdirectory named `UDxFencedProcessesJava.log`:

```
$ tail VMart/v_vmart_node0001_catalog/UDxLogs/UDxFencedProcesses.log
2012-12-12 10:23:47.649 [Java-2164] 0x01 UDx side process (Java) started
2012-12-12 10:23:47.871 [Java-2164] 0x0b [UserMessage] add2ints - Got values a=5 and b=6
2012-12-12 10:23:48.598 [Java-2164] 0x0c Exiting UDx side process
```

The SQL name of the UDx is added to the log message, along with the string `[UserMessage]` to mark the entry as a message added by a call to the `log()` method. These additions make it easier for you to filter the log to find the messages generated by your UDx.
Handling Cancel Requests

The query that calls your UDx can be canceled (usually, by the user pressing CTRL+C in vsql). When the calling query is canceled, Vertica begins a process of shutting down your UDx. Since UDTFs can perform lengthy and costly processing, the Vertica Java SDK defines several ways that Vertica attempts to signal UDTFs to terminate before it takes the step of killing the fenced-mode JVM process that is executing the UDTF. These attempts to signal the UDTF can help reduce the amount of CPU and memory that is wasted by having the UDx process continue processing after its results are no longer required.

When the user cancels a UDx, Vertica takes the following steps:

1. It sets the isCanceled property on UDTFs to true. Your processPartition methods can test this property to see if the function call has been canceled.
2. It calls UDTF's TransformFunction.cancel method. You should override this method to perform any shutdown tasks (such as killing threads).
3. It calls all types of UDx's destroy method. You should implement this method to free any resources your UDx has allocated.
4. It kills the JVM process running your UDx.

The topics in this section explain how your UDTF can use the cancel API.

Exiting When the Calling Query Has Been Canceled

Since User-Defined Transform Functions (UDTFs) often perform lengthy and CPU-intensive processing, it makes sense for them to terminate if the query that called them has been canceled. Exiting when the query has been canceled helps prevent wasting CPU cycles and memory on continued processing.

The TransformFunction class has a getter named .isCanceled that returns true if the calling query has been canceled. Your processPartition method can periodically check the value of this getter to determine if the query has been canceled, and exit if it has.

How often your processPartition function calls isCanceled depends on how much processing it performs on each row of data. Calling isCanceled does add overhead to your function, so you shouldn't call it too often. For transforms that do not perform lengthy processing, you could check for cancelation every 100 or 1000 rows. If your processPartition performs extensive processing for each row, you may want to check isCanceled every 10 or so rows.
The following code fragment shows how you could have the StringTokenizer UDTF example check whether its query has been canceled:

```java
public class CancelableTokenizeString extends TransformFunction {
    @Override
    public void processPartition(ServerInterface srvInterface,
                                PartitionReader inputReader,
                                PartitionWriter outputWriter)
        throws UdfException, DestroyInvocation {
        // Loop over all rows passed in this partition.
        int rowcount = 0; // maintain count of rows processed
        do {
            rowcount++; // Processing new row
            // Check for cancelation every 100 rows
            if (rowcount % 100 == 0) {
                // Check to see if Vertica marked this class as canceled
                if (this.isCanceled()) {
                    srvInterface.log("Got canceled! Exiting...");
                    return;
                }
            }
        } // Rest of the function here
        . . .
    }
}
```

This example checks for cancelation after processing 100 rows in the partition of data. If the query has been canceled, the example logs a message, then returns to the caller to exit the function.

**Note:** You need to strike a balance between adding overhead to your functions by calling isCanceled and having your functions waste CPU time by running after their query has been canceled (a rare event). For functions such as StringTokenizer which have a low overall processing cost, it usually does not make sense to test for cancelation. The cost of adding overhead to all function calls outweigh the amount of resources wasted by having the function run to completion or having its JVM process killed by Vertica on the rare occasions that its query is canceled.

### Overriding the Cancel Method

Your User-Defined Transform Function (UDTF) can override the TransformFunction.cancel method that Vertica calls if the query that called the function has been canceled. You should override this method to perform an orderly shutdown of any additional processing that your UDx spawned. For example, you can have your cancel method shut down threads that your UDTF has spawned or signal a third-party library that it needs to stop processing and exit. Your cancel method must leave your UDTF's function class ready to
be destroyed, since Vertica calls the UDx's destroy method after the cancel method has exited.

Notes

- If your UDTF does not override cancel, Vertica assumes your UDTF does not need to perform any special cancel processing and calls the function class's destroy method to have it free any resources.

- Your cancel method is called from a different thread than the thread running your UDx's processPartition function.

- The call to the cancel method is not synchronized in any way with your UDTF's processPartition method. If you need your processPartition function to exit before your cancel method performs some action (killing threads, for example) you need to have the two methods synchronize their actions.

- If your cancel method runs for too long, Vertica kills the JVM side process your UDx.

Adding Metadata to Java UDx Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on a Vertica Analytic Database cluster and lets third-party users of your function know who created the function. Your library's metadata appears in the USER_LIBRARIES system table after your library has been loaded into the Vertica Analytic Database catalog.

To add metadata to your Java UDx library, you create a subclass of the UDXLibrary class that contains your library's metadata. You then include this class within your JAR file. When you load your class into the Vertica Analytic Database catalog using the CREATE LIBRARY statement, looks for a subclass of UDXLibrary for the library's metadata.

In your subclass of UDXLibrary, you need to implement eight getters that return String values containing the library's metadata. The getters in this class are:

- getAuthor() returns the name you want associated with the creation of the library (your own name or your company's name for example).

- getLibraryBuildTag() returns whatever String you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when
the library was compiled). This is useful for tracking instances of your library as you are developing them.

- `getLibraryVersion()` returns the version of your library. You can use whatever numbering or naming scheme you want.

- `getLibrarySDKVersion()` returns the version of the Vertica Analytic Database SDK Library for which you've compiled the library.

Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytic Database server. The version of the Vertica Analytic Database SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytic Database server uses to determine if your library is compatible with it.

- `getSourceUrl()` returns a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.

- `getDescription()` returns a concise description of your library.

- `getLicensesRequired()` returns a placeholder for licensing information. You must pass an empty string for this value.

- `getSignature()` returns a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

For example, the following code demonstrates creating a UDXLibrary subclass to be included in the Add2Ints UDSF example JAR file (see Java Example: Add2Ints).

```java
// Import the UDXLibrary class to hold the metadata
import com.vertica.sdk.UDXLibrary;

public class Add2IntsLibrary extends UDXLibrary
{
    // Return values for the metadata about this library.
    @Override public String getAuthor() {return "Whizzo Analytics Ltd.";}
    @Override public String getLibraryBuildTag() {return "1234";}
    @Override public String getLibraryVersion() {return "1.0";}
    @Override public String getLibrarySDKVersion() {return "7.0.0";}
    @Override public String getSourceUrl() {
        return "http://example.com/add2ints";
    }
    @Override public String getDescription() {
        return "My Awesome Add 2 Ints Library";
    }
}
```
@Override
public
String
getLicensesRequired()
{
return "";
}

@Override
public
String
getSignature()
{
return "";
}

When the library containing the Add2IntsLibrary class loaded, the metadata appears in the USER_LIBRARIES system table:

```
=> CREATE LIBRARY JavaAdd2IntsLib AS :libfile LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE FUNCTION JavaAdd2Ints as LANGUAGE 'JAVA' name 'com.mycompany.example.Add2IntsFactory'
library JavaAdd2IntsLib;
CREATE FUNCTION
=> \x
Expanded display is on.
=> SELECT * FROM USER_LIBRARIES WHERE lib_name = 'JavaAdd2IntsLib';
- [ RECORD 1 ]-+---------------------------------------
schema_name | public
lib_name | JavaAdd2IntsLib
lib_oid | 45035996273869844
author | Whizzo Analytics Ltd.
owner_id | 45035996273704962
lib_file_name | public_JavaAdd2IntsLib_45035996273869844.jar
md5_sum | f3bfc76791daee95e4e2c0f8a8d2737f
sdk_version | v7.0.0-20131105
revision | 125200
lib_build_tag | 1234
lib_version | 1.0
lib_sdk_version | 7.0.0
source_url | http://example.com/add2ints
description | My Awesome Add 2 Ints Library
licenses_required | |
signature | |
```

Java UDx Resource Management

Java Virtual Machines (JVMs) allocate a set amount of memory when they start. This set memory allocation complicates memory management for Java UDxs, because memory cannot be dynamically allocated and freed by the UDx as it is processing data. This differs from C++ UDxs which can dynamically allocate resources.

To control the amount of memory consumed by Java UDxs, Vertica has a memory pool named jvm that it uses to allocate memory for JVMs. If this memory pool is exhausted, queries that call Java UDxs block until enough memory in the pool becomes free to start a new JVM.

By default, the jvm pool has:

- no memory of its own assigned to it, so it borrows memory from the GENERAL pool.
- its MAXMEMORYSIZE set to either 10% of system memory or 2GB, whichever is smaller.
• its PLANNEDCONCURRENCY set to AUTO, so that it inherits the GENERAL pool's PLANNEDCONCURRENCY setting.

You can view the current settings for the jvm pool by querying the RESOURCE_POOLS table:

```sql
=> SELECT MAXMEMORYSIZE, PLANNEDCONCURRENCY FROM V_CATALOG.RESOURCE_POOLS WHERE NAME = 'jvm';
MAXMEMORYSIZE | PLANNEDCONCURRENCY
--------------+-------------------
 10%          | AUTO
```

When a SQL statement calls a Java UDx, Vertica checks if the jvm memory pool has enough memory to start a new JVM instance to execute the function call. Vertica starts each new JVM with its heap memory size set to approximately the jvm pool's MAXMEMORYSIZE parameter divided by its PLANNEDCONCURRENCY parameter. If the memory pool does not contain enough memory, the query blocks until another JVM exits and return their memory to the pool.

If your Java UDx attempts to consume more memory than has been allocated to the JVM's heap size, it exits with a memory error. You can attempt to resolve this issue by:

• increasing the jvm pool’s MAXMEMORYSIZE parameter.
• decreasing the jvm pool’s PLANNEDCONCURRENCY parameter.
• changing your Java UDx’s code to consume less memory.

### Adjusting the jvm Pool

When adjusting the jvm pool to your needs, you must consider two factors:

• the amount of RAM your Java UDx requires to run
• how many concurrent Java UDx functions you expect your database to run

You can learn the amount of memory your Java UDx needs using several methods. For example, your code can use Java's `Runtime` class to get an estimate of the total memory it has allocated and then log the value using `ServerInterface.log()`. (An instance of this class is passed to your UDx.) If you have multiple Java UDxs in your database, set the jvm pool memory size based on the UDx that uses the most memory.

The number of concurrent sessions that need to run Java UDxs may not be the same as the global PLANNEDCONCURRENCY setting. For example, you may have just a single user who runs a Java UDx, which means you can lower the jvm pool’s PLANNEDCONCURRENCY setting to 1.
When you have an estimate for the amount of RAM and the number of concurrent user
sessions that need to run Java UDXs, you can adjust the jvm pool to an appropriate size. Set the
pool’s MAXMEMORYSIZE to the maximum amount of RAM needed by the most demanding
Java UDX multiplied by the number of concurrent user sessions that need to run Java UDXs. Set
the pool’s PLANNEDCONCURRENCY to the number of simultaneous user sessions that need to
run Java UDXs.

For example, suppose your Java UDX requires up to 4GB of memory to run and you expect up
to two user sessions use Java UDX's. You would use the following command to adjust the jvm
pool:

```
=> ALTER RESOURCE POOL jvm MAXMEMORYSIZE '8G' PLANNEDCONCURRENCY 2;
```

The MEMORYSIZE is set to 8GB, which is the 4GB maximum memory use by the Java UDX
multiplied by the 2 concurrent user sessions.

**Note:** The PLANNEDCONCURRENCY value is not the number of calls to Java UDX that you
expect to happen simultaneously. Instead, it is the number of concurrently open user
sessions that call Java UDXs at any time during the session. See below for more
information.

See [Managing Workloads](https://vertica.com/docs/Managing-Workloads.html) in the Administrator's Guide for more information on tuning the jvm and other resource pools.

### Freeing JVM Memory

The first time users call a Java UDX during their session, Vertica allocates memory from the jvm
pool and starts a new JVM. This JVM remains running for as long as the user's session is open
so it can process other Java UDX calls. Keeping the JVM running lowers the overhead of
executing multiple Java UDXs by the same session. If the JVM did not remain open, each call to
a Java UDX would require additional time for Vertica to allocate resources and start a new JVM.
However, having the JVM remain open means that the JVM's memory remains allocated for
the life of the session whether or not it will be used again.

If the jvm memory pool is depleted, queries containing Java UDXs either block until memory
becomes available or eventually fail due a lack of resources. If you find queries blocking or
failing for this reason, you can allocate more memory to the jvm pool and increase its
PLANNEDCONCURRENCY. Another option is to ask users to call the `RELEASE_JVM_MEMORY`
function when they no longer need to run Java UDXs. This function closes any JVM belonging to
the user's session and returns its allocated memory to the jvm memory pool.
The following example demonstrates querying V_MONITOR.SESSIONS to find the memory allocated to JVMs by all sessions. It also demonstrates how the memory is allocated by a call to a Java U Dx, and then freed by calling RELEASE_JVM_MEMORY.

```sql
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
user_name | external_memory_kb
----------|---------------------
dbadmin   | 0                   
(1 row)   

=> -- Call a Java U Dx
=> SELECT add2ints(123,456);  
add2ints
579       
(1 row)   

=> -- JVM is now running and memory is allocated to it.
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
USER_NAME | EXTERNAL_MEMORY_KB
----------|---------------------
dbadmin   | 79705               
(1 row)   

=> -- Shut down the JVM and deallocate memory
=> SELECT RELEASE_JVM_MEMORY();  
RELEASE_JVM_MEMORY
----------
Java process killed and memory released
(1 row)   

=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
USER_NAME | EXTERNAL_MEMORY_KB
----------|---------------------
dbadmin   | 0                   
(1 row)   
```

In rare cases, you may need to close all JVMs. For example, you may need to free memory for an important query, or several instances of a Java U Dx may be taking too long to complete. You can use the RELEASE_ALL_JVM_MEMORY to close all of the JVMs in all user sessions:

```sql
=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
USER_NAME | EXTERNAL_MEMORY_KB
----------|---------------------
ExampleUser| 79705               
  dbadmin  | 79705               
(2 rows)   

=> SELECT RELEASE_ALL_JVM_MEMORY();  
RELEASE_ALL_JVM_MEMORY
----------
Close all JVM sessions command sent. Check v_monitor.sessions for progress.
(1 row)   

=> SELECT USER_NAME,EXTERNAL_MEMORY_KB FROM V_MONITOR.SESSIONS;
USER_NAME | EXTERNAL_MEMORY_KB
----------|---------------------
dbadmin   | 0                   
(1 row)   
```
Caution: This function terminates all JVMs, including ones that are currently executing Java UDXs. This will cause any query that is currently executing a Java UDX to return an error.

Notes

- The jvm resource pool is used only to allocate memory for the Java UDX function calls in a statement. The rest of the resources required by the SQL statement come from other memory pools.

- The first time a Java UDX is called, Vertica starts a JVM to execute some Java methods to get metadata about the UDX during the query planning phase. The memory for this JVM is also taken from the jvm memory pool.

Developing with the Python SDK

The Vertica Python SDK provides new capabilities to the Vertica Analytic Database. Unlike the other Vertica Software Development Kits (SDKs), the Vertica Python SDK does not require any additional system configuration or header files. This low overhead allows you to develop and deploy new capabilities to your Vertica cluster in a short amount of time.

The following workflow is typical for the Python SDK:

Because Python is an interpreted language, you do not have to compile your program before loading the UDX in Vertica. However, you should expect to do some debugging of your code after you create your function and begin testing it in Vertica.

You can find detailed documentation of all of the classes in the Vertica Python SDK Documentation.
Python Libraries

Before you can use your Python UDx, you need to verify that it meets the following library requirements:

- Your UDx must import the `vertica_sdk` package in your code. You do not need to download this package. It is included as a part of the Vertica server.

```python
import vertica_sdk
```

- The Vertica Python SDK includes the Python Standard Library. Be aware that importing a package other than the Python Standard Library results in failure.

Python and Vertica Data Types

The Vertica Python SDK converts native Vertica data types into the appropriate Python data types, as follows:

<table>
<thead>
<tr>
<th>Vertica Data Type</th>
<th>Python Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>NUMERIC</td>
<td><code>decimal.Decimal</code></td>
</tr>
<tr>
<td>DATE</td>
<td><code>datetime.date</code></td>
</tr>
<tr>
<td>CHAR, VARCHAR, LONG VARCHAR</td>
<td>string (UTF-8 encoded)</td>
</tr>
<tr>
<td>BINARY, VARBINARY, LONG VARBINARY</td>
<td><code>binary</code></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td><code>datetime.datetime</code></td>
</tr>
<tr>
<td>TIME</td>
<td><code>datetime.time</code></td>
</tr>
</tbody>
</table>

*Note: Some Vertica Analytic Database data types are not supported in Python.*
Handling Errors

If your UDx encounters an unrecoverable error, it should throw a UdfException. The exception triggers a rollback of the function call. In Python, this function rollback occurs when the UDx raises an exception.

The following code shows how you can add error checking to your UDx. In this example, if one of the arguments is less than 100, then the Python UDx throws an error.

```python
while(True):
    # Example of error checking best practices.
    product_id = block_reader.getInt(2)
    if product_id < 100:
        raise ValueError("Invalid Product ID")
```

When an exception is raised in your Python UDx, the UDx throws a UdfException and generates an error message.

```sql
=> SELECT add2ints(prod_cost, sale_price, product_id) FROM bunch_of_numbers;
ERROR 3399: Failure in U Dx RPC call InvokeProcessBlock() : Error calling processBlock() in User Defined Object [add2ints]
at [/scratch_a/release/svrtar11244/vbuild/vertica/OSS/UDxFence/PythonInterface.cpp:168], error code: 0,
message: Error [/scratch_a/release/svrtar11244/vbuild/vertica/OSS/UDxFence/PythonInterface.cpp:385] function ['call_method']
(Python error type [class 'ValueError'])
Traceback (most recent call last):
  File "/home/dbadmin/py_db/v_py_db_node0001_catalog/Library/02fc4af0ace6f91eefa74baeef3ef76000a00000004fc4/pylib_02fc4af0ace6f91eefa74baeef3ef76000a00000004fc4.py", line 13, in processBlock
    raise ValueError("Invalid Product ID")
ValueError: Invalid Product ID
```

See Also

Writing Messages to Log Files

Writing Messages to Log Files

Writing messages to a log can help you when you debug your Python UDxs and want to output additional information about an error condition.

To write a message to the vertica log file, use the `server_interface.log()` function:
def processBlock(self, server_interface, arg_reader, res_writer):
    server_interface.log("Python UDx - Adding 2 ints!")
    while(True):
        first_int = block_reader.getInt(0)
        second_int = block_reader.getInt(1)
        block_writer.setInt(first_int + second_int)
        server_interface.log("Values: first_int is {} second_int is ".format(first_int, second_int))
    block_writer.next()
    if not block_reader.next():
        break

Vertica writes the messages to a log file stored in the catalog directory's UDxLog subdirectory, which is named UDxFencedProcesses.log:

$ tail /home/dbadmin/py_db/v_py_db_node0001_catalog/UDxLogs/UDxFencedProcesses.log
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780
PythonExecContext::processBlock
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780 [UserMessage] add2ints - Python UDx - Adding 2 ints!
07:52:12.862 [Python-v_py_db_node0001-7524:0x206c-40575] 0x7f70eee2f780 [UserMessage] add2ints - Values: first_int is 100 second_int is 100

Vertica adds the SQL name of the UDx to the log message. It also adds the string [UserMessage] to mark the entry as a message added by a call to the server_interface.log() function. These additions allow you to filter the log to find the messages generated by your UDx.

See Also
Handling Errors

Developing with the R SDK

The Vertica R SDK extends the capabilities of the Vertica Analytic Database so you can leverage additional R libraries. Before you can begin developing User Defined Extensions (UDxs) in R, you must install the R Language Pack for Vertica on each of the nodes in your cluster. The R SDK supports scalar and transform functions in fenced mode. Other UDx types are not supported.

The following workflow is typical for the R SDK:
You can find detailed documentation of all of the classes in the Vertica R SDK API Documentation.

Installing/Upgrading the R Language Pack for Vertica

To create R UDxs in Vertica, install the R Language Pack package that matches your server version. The R Language Pack includes the R runtime and associated libraries for interfacing with Vertica.

You must install the R Language Pack on each node in the cluster. The Vertica R Language Pack must be the only R Language Pack installed on the node.

Vertica R Language Pack Prerequisites

The R Language Pack package requires a number of packages for installation and execution. The names of these dependencies vary among Linux distributions. For Vertica supported Linux platforms the packages are:

- RHEL/CentOS - libfortran, xz-libs, libgomp
- SUSE Linux Enterprise Server - libfortran3, liblzma5, libgomp1
- Debian/Ubuntu - libfortran3, liblzma5, libgomp1

Installing the Vertica R Language Pack

If you use your operating systems package manager, rather than the rpm or dpkg command, for installation, you do not need to manually install the R Language Pack. The native package managers for each supported Linux version are:

- RHEL/CentOS - yum
- SUSE Linux Enterprise Server - zypper
- Debian/Ubuntu - apt-get
1. Download the R language package by going to the myVertica portal, clicking the downloads tab, and selecting the vertica-R-lang-_version_.rpm (or vertica-R-lang_<version>.deb) file for your server version. The R language package version must match your server version to three decimal points.

2. Install the package as root or using sudo:

   - **RHEL/CentOS**
     
     ```
     $ yum install vertica-R-lang_<version>.rpm
     ```

   - **SUSE Linux Enterprise Server**
     
     ```
     $ zypper install vertica-R-lang_<version>.rpm
     ```

   - **Debian**
     
     ```
     $ apt-get install ./vertica-R-lang_<version>.deb
     ```

   The installer puts the R binary in /opt/vertica/R.

### Upgrading the Vertica R Language Pack

When upgrading, some R packages you have manually installed may not work and may have to be reinstalled. If you do not update your package(s), then R returns an error if the package cannot be used. Instructions for upgrading these packages are below.

**Note:** The R packages provided in the R Language Pack are automatically upgraded and do not need to be reinstalled.

1. You must uninstall the R Language package before upgrading Vertica. Any additional R packages you manually installed remain in /opt/vertica/R and are not removed when you uninstall the package.

2. Upgrade your server package as detailed in Upgrading Vertica to a New Version.

3. After the server package has been updated, install the new R Language package on each host.

If you have installed additional R packages, on each node:
1. As root run `/opt/vertica/R/bin/R` and issue the command:

   ```
   > update.packages(checkBuilt=TRUE)
   ```

2. Select a CRAN mirror from the list displayed.

3. You are prompted to update each package that has an update available for it. You must update any packages that you manually installed and are not compatible with the current version of R in the R Language Pack. Do NOT update:

   - Rcpp
   - RInside

   The packages you selected to be updated are installed. Quit R with the command:

   ```
   > quit()
   ```

Vertica UDX functions written in R do not need to be compiled and you do not need to reload your Vertica-R libraries and functions after an upgrade.

**R Packages**

The Vertica R Language Pack includes the following R packages in addition to the default packages bundled with R:

- Rcpp
- RInside
- IpSolve
- IpSolveAPI

You can install additional R packages not included in the Vertica R Language Pack by using one of two methods. You must install the same packages on all nodes.

**Installing R Packages**

You can install additional R packages by using one of the two following methods.

Using the `install.packages()` R command:
$ sudo /opt/vertica/R/bin/R
> install.packages("Zelig");

Using CMD INSTALL:

```
/opt/vertica/R/bin/R CMD INSTALL <path-to-package-tgz>
```

The installed packages are located in: /opt/vertica/R/library.

**R and Vertica Data Types**

The following data types are supported when passing data to/from an R UDx:

<table>
<thead>
<tr>
<th>Vertica Data Type</th>
<th>R Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>logical</td>
</tr>
<tr>
<td>DATE, DATETIME, SMALLDATETIME, TIME, TIMESTAMP, TIMESTAMPZ, TIMETZ</td>
<td>numeric</td>
</tr>
<tr>
<td>DOUBLE PRECISION, FLOAT, REAL</td>
<td>numeric</td>
</tr>
<tr>
<td>BIGINT, DECIMAL, INT, NUMERIC, NUMBER, MONEY</td>
<td>numeric</td>
</tr>
<tr>
<td>BINARY, VARBINARY</td>
<td>character</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>character</td>
</tr>
</tbody>
</table>

NULL values in Vertica are translated to R NA values when sent to the R function. R NA values are translated into Vertica null values when returned from the R function to Vertica.

**Important:** When specifying LONG VARCHAR or LONG VARBINARY data types, include the space between the two words. For example, `datatype = c("long varchar")`.

**Handling Errors**

If your UDx encounters an unrecoverable error, it should throw a UdfException. The error triggers a rollback of the function call. In R, this function rollback occurs when the UDx executes an error action.

The following code shows how you can add error checking to your UDx. In this example, if the third column of the data frame does not match the specified Product ID, then the R UDx throws an error.
When an exception is raised in your R UDx, the UDx throws a UdfException and generates an error message.

```sql
=> SELECT Calculate_Cost_w_Tax(item_price, tax_rate, prod_id) FROM Inventory_Sales_Data;
```

Error calling processBlock() in User Defined Object [mul] at
`[/scratch_a/release/svrta30318/vbuild/vertica/OSS/UDxFence/RInterface.cpp:1308]`,
error code: 0, message: Exception in processBlockForR :Invalid Product ID!

### Adding Metadata to R Libraries

You can add metadata, such as author name, the version of the library, a description of your library, and so on to your library. This metadata lets you track the version of your function that is deployed on a Vertica Analytic Database cluster and lets third-party users of your function know who created the function. Your library's metadata appears in the `USER_LIBRARIES` system table after your library has been loaded into the Vertica Analytic Database catalog.

You declare the metadata for your library by calling the `RegisterLibrary()` function in one of the source files for your UDx. If there is more than one function call in the source files for your UDx, whichever gets interpreted last as Vertica Analytic Database loads the library is used to determine the library's metadata.

The `RegisterLibrary()` function takes eight string parameters:

```r
RegisterLibrary(author,
    library_build_tag,
    library_version,
    library_sdk_version,
    source_url,
    description,
    licenses_required,
    signature);
```
- **author** contains whatever name you want associated with the creation of the library (your own name or your company's name for example).

- **library_build_tag** is a string you want to use to represent the specific build of the library (for example, the SVN revision number or a timestamp of when the library was compiled). This is useful for tracking instances of your library as you are developing them.

- **library_version** is the version of your library. You can use whatever numbering or naming scheme you want.

- **library_sdk_version** is the version of the Vertica Analytic Database SDK Library for which you've compiled the library.

  Note: This field isn't used to determine whether a library is compatible with a version of the Vertica Analytic Database server. The version of the Vertica Analytic Database SDK you use to compile your library is embedded in the library when you compile it. It is this information that Vertica Analytic Database server uses to determine if your library is compatible with it.

- **source_url** is a URL where users of your function can find more information about it. This can be your company's website, the GitHub page hosting your library's source code, or whatever site you like.

- **description** is a concise description of your library.

- **licenses_required** is a placeholder for licensing information. You must pass an empty string for this value.

- **signature** is a placeholder for a signature that will authenticate your library. You must pass an empty string for this value.

The following example shows how to add metadata to an R UDx.

```python
RegisterLibrary("Speedy Analytics Ltd.",
 "1234",
 "1.0",
 "8.1.0",
 "Sales Tax R Library",
 "",
 "")
```

Loading the library and querying the USER_LIBRARIES system table shows the metadata supplied in the call to RegisterLibrary:
Setting Null Input and Volatility Behavior for R Functions

Vertica supports defining volatility and null-input settings for UDxs written in R. Both settings aid in the performance of your R function.

Volatility Settings

Volatility settings describe the behavior of the function to the Vertica optimizer. For example, if you have identical rows of input data and you know the UDx is immutable, then you can define the UDx as IMMUTABLE. This tells the Vertica optimizer that it can return a cached value for subsequent identical rows on which the function is called rather than having the function run on each identical row.

To indicate your UDx's volatility, set the volatility parameter of your R factory function to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLATILE</td>
<td>Repeated calls to the function with the same arguments always result in different values. Vertica always calls volatile functions for each invocation.</td>
</tr>
<tr>
<td>IMMUTABLE</td>
<td>Calls to the function with the same arguments always results in the same return value.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STABLE</td>
<td>Repeated calls to the function with the same arguments <em>within the same statement</em> returns the same output. For example, a function that returns the current user name is stable because the user cannot change within a statement. The user name could change between statements.</td>
</tr>
<tr>
<td>DEFAULT_VOLATILITY</td>
<td>The default volatility. This is the same as VOLATILE.</td>
</tr>
</tbody>
</table>

If you do not define a volatility, then the function is considered to be VOLATILE.

The following example sets the volatility to STABLE in the `multiplyTwoIntsFactory` function:

```r
multiplyTwoIntsFactory <- function() {
  list(name = multiplyTwoInts,
       udxtype = c("scalar"),
       intype = c("float","float"),
       outtype = c("float"),
       volatility = c("stable"),
       parametertypecallback = multiplyTwoIntsParameters)
}
```

**Null Input Behavior**

Null input setting determine how to respond to rows that have null input. For example, you can choose to return null if any inputs are null rather than calling the function and having the function deal with a NULL input.

To indicate how your UDx reacts to NULL input, set the strictness parameter of your R factory function to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLED_ON_NULL_INPUT</td>
<td>The function must be called, even if one or more arguments are NULL.</td>
</tr>
<tr>
<td>RETURN_NULL_ON_NULL_INPUT</td>
<td>The function always returns a NULL value if any of its arguments are NULL.</td>
</tr>
<tr>
<td>STRICT</td>
<td>A synonym for RETURN_NULL_ON_NULL_INPUT</td>
</tr>
<tr>
<td>DEFAULT STRICTNESS</td>
<td>The default strictness setting. This is the same as CALLED_ON_NULL_INPUT.</td>
</tr>
</tbody>
</table>
If you do not define a null input behavior, then the function is called on every row of data regardless of the presence of NULL values.

The following example sets the NULL input behavior to STRICT in the multiplyTwoIntsFactory function:

```r
multiplyTwoIntsFactory <- function() {
  list(name = multiplyTwoInts,
       udxtype = c("scalar"),
       intype = c("float","float"),
       outtype = c("float"),
       strictness = c("strict"),
       parametertypecallback = multiplyTwoIntsParameters)
}
```
Aggregate Functions (UDAFs)

Aggregate functions perform an operation on a set of values and return one value. Vertica provides standard built-in aggregate functions such as **AVG**, **MAX**, and **MIN**. User-Defined Aggregate Functions work similarly to the built-in aggregate functions.

User-Defined Aggregate Functions:

- Support a single input column (or set) of values and provide a single output column.
- Support RLE decompression; RLE input is decompressed before it is sent to a UDAF.
- Can be used with the GROUP BY and HAVING clauses. Only columns appearing in the GROUP BY clause can be selected.
- Cannot be used with correlated subquery.

UDAFs are available for C++ only.

UDAF Class Overview

You create your UDAF by subclassing two classes defined by the Vertica SDK: `AggregateFunction` and `AggregateFunctionFactory`.

AggregateFunction

The `AggregateFunction` class performs the aggregation. It computes values on each database node where relevant data is stored and then combines the results from the nodes. You must implement the following methods:

- `initAggregate()` - Initializes the class, defines variables, and sets the starting value for the variables. This function must be idempotent.
- `aggregate()` - The main aggregation operation, executed on each node.
- `combine()` - If multiple invocations of `aggregate()` are needed, Vertica calls `combine()` to combine all the sub-aggregations into a final aggregation. Although this method might
not be called, you must define it.

- **terminate()** - Terminates the function and returns the result as a column.

**Important:** The `aggregate()` function might not operate on the complete input set all at once. For this reason, `initAggregate()` must be idempotent.

The `AggregateFunction` class also provides optional methods that you can implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDAF API (see Allocating Resources for UDxs for details).

### AggregateFunctionFactory

The `AggregateFunctionFactory` class specifies metadata information such as the argument and return types of your aggregate function. It also instantiates your `AggregateFunction` subclass. Your subclass must implement the following methods:

- **getPrototype()** - Defines the number of parameters and data types accepted by the function. There is a single parameter for aggregate functions.

- **getIntermediateTypes()** - Defines the intermediate variable(s) used by the function. These variables are used when combining the results of `aggregate()` calls.

- **getParameterType()** - Defines the names and types of parameters that this function uses (optional).

- **getReturnType()** - Defines the type of the output column.

Vertica uses this data when you call the CREATE AGGREGATE FUNCTION SQL statement to add the function to the database catalog.

### UDAF Performance in Statements Containing a GROUP BY Clause

You may see slower-than-expected performance from your UDAF if the SQL statement calling it also contains a GROUP BY Clause. For example:

```sql
=> SELECT a, MYUDAF(b) FROM sampletable GROUP BY a;
```
In statements like this one, Vertica does not consolidate row data together before calling your UDAF's `aggregate()` method. Instead, it calls `aggregate()` once for each row of data. Usually, the overhead of having Vertica consolidate the row data is greater than the overhead of calling `aggregate()` for each row of data. However, if your UDAF's `aggregate()` method has significant overhead, then you might notice an impact on your UDAF's performance.

For example, suppose `aggregate()` allocates memory. When called in a statement with a GROUP BY clause, it performs this memory allocation for each row of data. Because memory allocation is a relatively expensive process, this allocation can impact the overall performance of your UDAF and the query.

There are two ways you can address UDAF performance in a statement containing a GROUP BY clause:

- Reduce the overhead of each call to `aggregate()`. If possible, move any allocation or other setup operations to the UDAF's `setup()` function.

- Declare a special parameter that tells Vertica to group row data together when calling a UDAF. This technique is explained below.

### Using the `_minimizeCallCount` Parameter

Your UDAF can tell Vertica to always batch row data together to reduce the number of calls to its `aggregate()` method. To trigger this behavior, your UDAF must declare an integer parameter named `_minimizeCallCount`. You do not need to set a value for this parameter in your SQL statement. The fact that your UDAF declares this parameter triggers Vertica to group row data together when calling `aggregate()`.

You declare the `_minimizeCallCount` parameter the same way you declare other UDx parameters. See UDx Parameters for more information.

**Important:** Always test the performance of your UDAF before and after implementing the `_minimizeCallCount` parameter to ensure that it improves performance. You might find that the overhead of having Vertica group row data for your UDAF is greater than the cost of the repeated calls to `aggregate()`.

### C++ API

This section provides APIs and examples for the C++ API for UDAFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see Developing with the C++ SDK.
AggregateFunction and AggregateFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See UDAF Class Overview for general information about implementing the AggregateFunction and AggregateFunctionFactory classes.

AggregateFunction API

The API provides the following methods for extension by subclasses:

```cpp
virtual void setup(ServerInterface &srvInterface,
                    const SizedColumnTypes &argTypes);

virtual void initAggregate(ServerInterface &srvInterface, IntermediateAggs &aggs)=0;

void aggregate(ServerInterface &srvInterface, BlockReader &arg_reader,
                 IntermediateAggs &aggs);

virtual void combine(ServerInterface &srvInterface, IntermediateAggs &aggs_output,
                      MultipleIntermediateAggs &aggs_other)=0;

virtual void terminate(ServerInterface &srvInterface, BlockWriter &res_writer,
                        IntermediateAggs &aggs);

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);
```

AggregateFunctionFactory API

The API provides the following methods for extension by subclasses:

```cpp
virtual AggregateFunction *
     createAggregateFunction ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
                            ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getIntermediateTypes(ServerInterface &srvInterface,
                                   const SizedColumnTypes &inputTypes,
                                   SizedColumnTypes &intermediateTypeMetaData)=0;

virtual void getReturnType(ServerInterface &srvInterface,
                            const SizedColumnTypes &argTypes,
                            SizedColumnTypes &returnType)=0;

virtual void getParameterType(ServerInterface &srvInterface,
                               SizedColumnTypes &parameterTypes);
```
C++ Example: Average

The Average aggregate function created in this example computes the average of values in a column.

You can find the source code used in this example on the Vertica GitHub page.

Loading the Example

Use CREATE LIBRARY and CREATE AGGREGATE FUNCTION to declare the function:

```sql
=> CREATE LIBRARY AggregateFunctions AS '/opt/vertica/sdk/examples/build/AggregateFunctions.so';
CREATE LIBRARY
=> CREATE aggregate function ag_avg AS LANGUAGE 'C++'
name 'AverageFactory' library AggregateFunctions;
CREATE AGGREGATE FUNCTION
```

Using the Example

Use the function as part of a SELECT statement:

```sql
=> SELECT * FROM average;
  id | count
---+------
 A |  8
 B |  3
 C |  6
 D |  2
 E |  9
 F |  7
 G |  5
 H |  4
 I |  1
(9 rows)
=> SELECT ag_avg(count) FROM average;
 ag_avg
-------
   5
(1 row)
```

AggregateFunction Implementation

This example adds the input argument values in the aggregate() method and keeps a counter of the number of values added. The server runs aggregate() on every node and different data chunks, and combines all the individually added values and counters in the
combine() method. Finally, the average value is computed in the terminate() method by dividing the total sum by the total number of values processed.

For this discussion, assume the following environment:

- A three-node Vertica cluster
- A table column that contains nine values that are evenly distributed across the nodes. Schematically, the nodes look like the following figure:
Node 1
Values 1,2,3

Node 2
Values 4,5,6

Node 3
Values 7,8,9
The function uses sum and count variables. *Sum* contains the sum of the values, and *count* contains the count of values.

First, `initAggregate()` initializes the variables and sets their values to zero.

```cpp
virtual void initAggregate(ServerInterface &srvInterface,
                           IntermediateAggs &aggs)
{
  try {
    VNumeric &sum = aggs.getNumericRef(0);
    sum.setZero();

    vint &count = aggs.getIntRef(1);
    count = 0;
  } catch(std::exception &e) {
    vt_report_error(0, "Exception while initializing intermediate aggregates: [% s]", e.what());
  }
}
```

The aggregate() function reads the block of data on each node and calculates partial aggregates.

```cpp
void aggregate(ServerInterface &srvInterface,
               BlockReader &argReader,
               IntermediateAggs &aggs)
{
  try {
    VNumeric &sum = aggs.getNumericRef(0);
    vint &count = aggs.getIntRef(1);

    do {
      const VNumeric &input = argReader.getNumericRef(0);
      if (!input.isNull()) {
        sum.accumulate(&input);
        count++;
      }
    } while (argReader.next());
  } catch(std::exception &e) {
    vt_report_error(0, " Exception while processing aggregate: [% s]", e.what());
  }
}
```

Each completed instance of the `aggregate()` function returns multiple partial aggregates for *sum* and *count*. The following figure illustrates this process using the `aggregate()` function:
Node 1
Values 1,2,3

aggregate()
sum = 6; count = 3

initAggregate()
sum = 0; count = 0

Node 2
Values 4,5,6

aggregate()
sum = 15; count = 3

Node 3
Values 7,8,9

aggregate()
sum = 24; count = 3
The **combine()** function puts together the partial aggregates calculated by each instance of the average function.

```cpp
virtual void combine(ServerInterface &srvInterface,
                      IntermediateAggs &aggs,
                      MultipleIntermediateAggs &aggsOther)
{
    try {
        VNumeric &mySum = aggs.getNumericRef(0);
        vint &myCount = aggs.getIntRef(1);
        do {
            const VNumeric &otherSum = aggsOther.getNumericRef(0);
            const vint &otherCount = aggsOther.getIntRef(1);
            mySum.accumulate(&otherSum);
            myCount += otherCount;
        } while (aggsOther.next());
    } catch(std::exception &e) {
        vt_report_error(0, "Exception while combining intermediate aggregates: [%s]", e.what());
    }
}
```

The following figure shows how each partial aggregate is combined:
After all input has been evaluated by the `aggregate()` function Vertica calls the `terminate()` function. It returns the average to the caller.
virtual void terminate(ServerInterface &srvInterface,
                      BlockWriter &resWriter,
                      IntermediateAggs &aggs)
{
    try {
        const int32 MAX_INT_PRECISION = 20;
        const int32 prec = Basics::getNumericWordCount(MAX_INT_PRECISION);
        uint64 words[prec];
        VNumeric count(words, prec, 0/*scale*/);
        count.copy(aggs.getIntRef(1));
        VNumeric &out = resWriter.getNumericRef();
        if (count.isZero()) {
            out.setNull();
        } else {
            const VNumeric &sum = aggs.getNumericRef(0);
            out.div(&sum, &count);
        }
    }
}

The following figure shows the implementation of the terminate() function:

![Diagram]

Vertica Analytic Database (9.0.x)
AggregateFunctionFactory Implementation

The `getPrototype()` function allows you to define the variables that are sent to your aggregate function and returned to Vertica after your aggregate function runs. The following example accepts and returns a numeric value:

```cpp
virtual void getPrototype(ServerInterface &srvfloaterface, 
ColumnTypes &argTypes, 
ColumnTypes &returnType)
{
    argTypes.addNumeric();
    returnType.addNumeric();
}
```

The `getIntermediateTypes()` function defines any intermediate variables that you use in your aggregate function. *Intermediate variables* are values used to pass data among multiple invocations of an aggregate function. They are used to combine results until a final result can be computed. In this example, there are two results - total (numeric) and count (int).

```cpp
virtual void getIntermediateTypes(ServerInterface &srvInterface, 
const SizedColumnTypes &inputTypes, 
SizedColumnTypes &intermediateTypeMetaData)
{
    const VerticaType &inType = inputTypes.getColumnType(0);
    intermediateTypeMetaData.addNumeric(inType.getNumericScale());
    intermediateTypeMetaData.addInt();
}
```

The `getReturnType()` function defines the output data type:

```cpp
virtual void getReturnType(ServerInterface &srvfloaterface, 
const SizedColumnTypes &inputTypes, 
SizedColumnTypes &outputTypes)
{
    const VerticaType &inType = inputTypes.getColumnType(0);
    outputTypes.addNumeric(inType.getNumericPrecision(),
    inType.getNumericScale());
}
```

Analytic Functions (UDAnFs)

User-Defined Analytic Functions (UDAnFs) are used for analytics. See SQL Analytics for an overview of Vertica's built-in analytics. Like User-Defined Scalar Functions (UDSFs), UDAnFs
must output a single value for each row of data read and can have no more than 1600 arguments.

Unlike UDSFs, the UDAnF’s input reader and output reader can be advanced independently. This feature lets you create UDAnF’s where the output value is calculated over multiple rows of data. By advancing the reader and writer independently, you can create functions similar to the built-in analytic functions such as LAG, which uses data from prior rows to output a value for the current row.

**UDAnF Class Overview**

You create your UDAnF by subclassing two classes defined by the Vertica SDK: AnalyticFunction and AnalyticFunctionFactory.

**AnalyticFunction**

The AnalyticFunction subclass performs the analytic processing. Your subclass must define the processPartition() method to perform the operation. It may define methods to set up and tear down the function.

**Performing the Operation**

The processPartition() method reads a partition of data, performs some sort of processing, and outputs a single value for each input row.

Vertica calls processPartition() once for each partition of data. It supplies the partition using an AnalyticPartitionReader object from which you read its input data. In addition, there is a unique method on this object named isNewOrderByKey(), which returns a Boolean value indicating whether your function has seen a row with the same ORDER BY key (or keys). This method is very useful for analytic functions (such as the example RANK function) which need to handle rows with identical ORDER BY keys differently than rows with different ORDER BY keys.

**Note:** You can specify multiple ORDER BY columns in the SQL query you use to call your UDAnF. The isNewOrderByKey method returns true if any of the ORDER BY keys are different than the previous row.

Once your method has finished processing the row of data, you advance it to the next row of input by calling next() on AnalyticPartitionReader.

Your method writes its output value using an AnalyticPartitionWriter object that Vertica supplies as a parameter to processPartition(). This object has data-type-specific
methods to write the output value (such as `setInt()`). After setting the output value, call `next()` on `AnalyticPartitionWriter` to advance to the next row in the output.

**Note:** You must be sure that your function produces a row of output for each row of input in the partition. You must also not output more rows than are in the partition, otherwise the zygote size process (if running in Fenced Mode) or Vertica itself could generate an out of bounds error.

**Setting Up and Tearing Down**

The `AnalyticFunction` class defines two additional methods that you can optionally implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDx API (see Allocating Resources for UDxs for details).

**AnalyticFunctionFactory**

The `AnalyticFunctionFactory` class tells Vertica metadata about your UDAnF: its number of parameters and their data types, as well as the data type of its return value. It also instantiates a subclass of `AnalyticFunction`.

Your `AnalyticFunctionFactory` subclass must implement the following methods:

- `getPrototype()` describes the input parameters and output value of your function. You set these values by calling functions on two `ColumnType` objects that are passed to your method.

- `createAnalyticFunction()` supplies an instance of your `AnalyticFunction` that Vertica can call to process a UDAnF function call.

- `getReturnType()` provides details about your function's output. This method is where you set the width of the output value if your function returns a variable-width value (such as VARCHAR) or the precision of the output value if it has a settable precision (such as TIMESTAMP).

**C++ API**

This section provides APIs and examples for the C++ API for UDAnFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see Developing with the C++ SDK.
AnalyticFunction and AnalyticFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See UDAf Class Overview for general information about implementing the AnalyticFunction and AnalyticFunctionFactory classes.

AnalyticFunction API

The API provides the following methods for extension by subclasses:

```cpp
virtual void setup(ServerInterface &srvInterface, 
                    const SizedColumnTypes &argTypes);

virtual void processPartition (ServerInterface &srvInterface, 
                                AnalyticPartitionReader &input_reader, 
                                AnalyticPartitionWriter &output_writer)=0;

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);
```

AnalyticFunctionFactory API

The API provides the following methods for extension by subclasses:

```cpp
virtual AnalyticFunction * createAnalyticFunction (ServerInterface &srvInterface)=0;

virtual void getPrototype (ServerInterface &srvInterface, 
                           ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getReturnType (ServerInterface &srvInterface, 
                            const SizedColumnTypes &argTypes, SizedColumnTypes &returnType)=0;

virtual void getParameterType (ServerInterface &srvInterface, 
                               SizedColumnTypes &parameterTypes);
```

C++ Example: Rank

The Rank analytic function ranks rows based on how they are ordered.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the AnalyticFunctions.so library that contains the function has been copied to the dbadmin
user's home directory on the initiator node.

```sql
=> CREATE LIBRARY AnalyticFunctions AS '/home/dbadmin/AnalyticFunctions.so';
CREATE LIBRARY
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'C++'
   NAME 'RankFactory' LIBRARY AnalyticFunctions;
CREATE ANALYTIC FUNCTION
```

An example of running this rank function, named `an_rank`, is:

```sql
=> SELECT * FROM hits;
site | date | num_hits
----|-----|--------
www.example.com | 2012-01-02 | 97
www.vertica.com | 2012-01-01 | 343435
www.example.com | 2012-01-01 | 123
www.example.com | 2012-01-04 | 112
www.vertica.com | 2012-01-02 | 503695
www.vertica.com | 2012-01-03 | 490387
www.example.com | 2012-01-03 | 123
(7 rows)

=> SELECT site, date, num_hits, an_rank()
   OVER (PARTITION BY site ORDER BY num_hits DESC)
   AS an_rank FROM hits;
   site | date | num_hits | an_rank
----|-----|---------|--------
www.example.com | 2012-01-03 | 123 | 1
www.example.com | 2012-01-01 | 123 | 1
www.example.com | 2012-01-04 | 112 | 3
www.example.com | 2012-01-02 | 97 | 4
www.vertica.com | 2012-01-02 | 503695 | 1
www.vertica.com | 2012-01-03 | 490387 | 2
www.vertica.com | 2012-01-01 | 343435 | 3
(7 rows)
```

As with the built-in `RANK` analytic function, rows that have the same value for the ORDER BY column (num_hits in this example) have the same rank, but the rank continues to increase, so that the next row that has a different ORDER BY key gets a rank value based on the number of rows that preceded it.

**AnalyticFunction Implementation**

The following code defines an AnalyticFunction subclass named Rank. It is based on example code distributed in the examples directory of the SDK.

```cpp
/**
 * User-defined analytic function: Rank - works mostly the same as SQL-99 rank
 * with the ability to define as many order by columns as desired
 */
class Rank : public AnalyticFunction
{
    virtual void processPartition(ServerInterface &srvInterface,
                                AnalyticPartitionReader &inputReader,
```
In this example, the `processPartition()` method does not actually read any of the data from the input row; it just advances through the rows. It does not need to read data; it just counts the rows that have been read and determine whether those rows have the same ORDER BY key as the previous row. If the current row is a new ORDER BY key, then the rank is set to the total number of rows that have been processed. If the current row has the same ORDER BY value as the previous row, then the rank remains the same.

Note that the function has a top-level try-catch block. All of your UDx functions should always have one to prevent stray exceptions from being passed back to Vertica (if you run the function unfenced) or the side process.

**AnalyticFunctionFactory Implementation**

The following code defines the AnalyticFunctionFactory that corresponds with the Rank analytic function.

```cpp
class RankFactory : public AnalyticFunctionFactory
{
    virtual void getPrototype(ServerInterface &srvInterface,
                               ColumnTypes &argTypes, ColumnTypes &returnType)
    {
        returnType.addInt();
    }
};
```
virtual void getReturnType(ServerInterface &srvInterface,
                         const SizedColumnTypes &inputTypes,
                         SizedColumnTypes &outputTypes)
{
    outputTypes.addInt();
}
virtual AnalyticFunction *createAnalyticFunction(ServerInterface &srvInterface)
{
    return vt_createFuncObj(srvInterface.allocator, Rank);
};

The first method defined by the RankFactory subclass, getPrototype(), sets the data type of the return value. Because the Rank UDAnF does not read input, it does not define any arguments by calling methods on the ColumnTypes object passed in the argTypes parameter.

The next method is getReturnType(). If your function returns a data type that needs to define a width or precision, your implementation of the getReturnType() method calls a method on the SizedColumnType object passed in as a parameter to tell Vertica the width or precision. Rank returns a fixed-width data type (an INTEGER) so it does not need to set the precision or width of its output; it just calls addInt() to report its output data type.

Finally, RankFactory defines the createAnalyticFunction() method that returns an instance of the AnalyticFunction class that Vertica can call. This code is mostly boilerplate. All you need to do is add the name of your analytic function class in the call to vt_createFuncObj(), which takes care of allocating the object for you.

Java API

This section provides APIs and examples for the Java API for UDAnFs.

For information on setting up a Java development environment and compiling and packaging libraries, see Developing with the Java SDK.

AnalyticFunction and AnalyticFunctionFactory Java Interface

This section describes information that is specific to the Java API. See UDAnF Class Overview for general information about implementing the AnalyticFunction and AnalyticFunctionFactory classes.

AnalyticFunction API

The API provides the following methods for extension by subclasses:
void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);

abstract void processPartition (ServerInterface srvInterface,
                               AnalyticPartitionReader input_reader, AnalyticPartitionWriter output_writer)
                               throws UdfException, DestroyInvocation;

void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);

AnalyticFunctionFactory API

The API provides the following methods for extension by subclasses:

abstract AnalyticFunction createAnalyticFunction (ServerInterface srvInterface);

abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);

abstract void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes,
                             SizedColumnTypes returnType) throws UdfException;

void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);

Java Example: Rank

The Rank analytic function ranks rows based on how they are ordered.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the AnalyticFunctions.jar library that contains the function has been copied to the dbadmin user's home directory on the initiator node.

=> CREATE LIBRARY AnalyticFunctions AS '/home/dbadmin/AnalyticFunctions.jar';
CREATE LIBRARY
=> CREATE ANALYTIC FUNCTION an_rank AS LANGUAGE 'Java'
     NAME 'RankFactory' LIBRARY AnalyticFunctions;
CREATE ANALYTIC FUNCTION

An example of running this rank function, named an_rank, is:

=> SELECT * FROM hits;
  site     | date      | num_hits
-----------+-----------+--------
www.example.com | 2012-01-02  | 97
www.vertica.com  | 2012-01-01  | 343435
www.example.com  | 2012-01-01  | 123
As with the built-in **RANK** analytic function, rows that have the same value for the ORDER BY column (num_hits in this example) have the same rank, but the rank continues to increase, so that the next row that has a different ORDER BY key gets a rank value based on the number of rows that preceded it.

**AnalyticFunction Implementation**

The following code defines an **AnalyticFunction** subclass named **Rank**. The code can be found in the SDK examples directory.

```java
/**
 * User-defined analytic function: Rank - works mostly the same as SQL-99 rank
 * with the ability to define as many order by columns as desired
 * *
 * public class Rank extends AnalyticFunction {
 *     private int rank, numRowsWithSameOrder;
 * 
 *     @Override
 *     public void processPartition(ServerInterface srvInterface,
 *                                  AnalyticPartitionReader inputReader,
 *                                  AnalyticPartitionWriter outputWriter)
 *         throws UdfException, DestroyInvocation {
 *         rank = 0;
 *         numRowsWithSameOrder = 1;
 *         do{
 *             if(!inputReader.isNewOrderByKey()){++numRowsWithSameOrder;
 *         }else {
 *             rank += numRowsWithSameOrder;
 *             numRowsWithSameOrder = 1;
 *         }
 *         outputWriter.setLong(0, rank);
 *         outputWriter.next();
 *     }while(inputReader.next());
 */
```
In this example, the processPartition() method does not actually read any of the data from the input row; it just advances through the rows. It just needs to count the number of rows that have been read and determine whether those rows have the same ORDER BY key as the previous row. If the current row is a new ORDER BY key, then the rank is set to the total number of rows that have been processed. If the current row has the same ORDER BY value as the previous row, then the rank remains the same.

AnalyticFunctionFactory Implementation

The following code defines the AnalyticFunctionFactory that corresponds with the Rank analytic function.

```java
public class RankFactory extends AnalyticFunctionFactory {

@Override
public void getPrototype(ServerInterface srvInterface,
   ColumnTypes argTypes,
   ColumnTypes returnType) {
   returnType.addInt();
}

@Override
public void getReturnType(ServerInterface srvInterface,
   SizedColumnTypes argTypes,
   SizedColumnTypes returnType) throws UdfException {
   returnType.addInt();
}

@Override
public AnalyticFunction createAnalyticFunction(ServerInterface srvInterface) {
   return new Rank();
}
}
```

The first method defined by the RankFactory subclass, getPrototype(), sets the data type of the return value. Because the Rank UDAf does not read input, it does not define any arguments by calling methods on the ColumnTypes object passed in the argTypes parameter.

The next method is getReturnType(). If your analytic function returns a data type that needs to define a width or precision, your implementation of the getReturnType function calls a method on the SizedColumnType parameter to tell Vertica the width or precision. Rank returns a fixed-width data type (an INTEGER) so it does not need to set the precision or width of its output; it just calls addInt() to report its output data type.

Finally, RankFactory defines the createAnalyticFunction() method that returns an instance of the AnalyticFunction class that Vertica can call.
Scalar Functions (UDSFs)

A User-Defined Scalar Function (UDSF) returns a single value for each row of data it reads. You can use a UDSF anywhere you can use a built-in Vertica function. You usually develop a UDSF to perform data manipulations that are too complex or too slow to perform using SQL statements and functions. UDSFs also let you use analytic functions provided by third-party libraries within Vertica while still maintaining high performance.

Your UDSF must return a value for every input row (unless it generates an error; see Handling Errors for details). Failing to return a value for a row will result in incorrect results and potentially destabilize the Vertica server if not run in Fenced Mode.

A UDSF cannot have more than 1600 arguments.

UDSF Class Overview

You create your UDSF by subclassing two classes defined by the Vertica SDK: ScalarFunction and ScalarFunctionFactory.

ScalarFunction

The ScalarFunction class is the heart of a UDSF. Your subclass must define the processBlock() method to perform the scalar operation. It may define methods to set up and tear down the function.

Performing the Operation

The processBlock() method carries out all of the processing that you want your UDSF to perform. When a user calls your function in a SQL statement, Vertica bundles together the data from the function parameters and passes it to processBlock().

The input and output of the processBlock() method are supplied by objects of the BlockReader and BlockWriter classes. They define methods that you use to read the input data and write the output data for your UDSF.

The majority of the work in developing a UDSF is writing processBlock(). This is where all of the processing in your function occurs. Your UDSF should follow this basic pattern:
- Read in a set of parameters from the BlockReader object using data-type-specific methods.
- Process the data in some manner.
- Output the resulting value using one of the BlockWriter class's data-type-specific methods.
- Advance to the next row of output and input by calling BlockWriter.next() and BlockReader.next().

This process continues until there are no more rows of data to be read (BlockReader.next() returns false).

You must make sure that processBlock() reads all of the rows in its input and outputs a single value for each row. Failure to do so can corrupt the data structures that Vertica reads to get the output of your UDSF. The only exception to this rule is if your processBlock() function reports an error back to Vertica (see Handling Errors). In that case, Vertica does not attempt to read the incomplete result set generated by the UDSF.

Setting Up and Tearing Down

The ScalarFunction class defines two additional methods that you can optionally implement to allocate and free resources: setup() and destroy(). You should use these methods to allocate and deallocate resources that you do not allocate through the UDx API (see Allocating Resources for UDxs for details).

Notes

- While the name you choose for your ScalarFunction subclass does not have to match the name of the SQL function you will later assign to it, Vertica considers making the names the same a best practice.
- Do not assume that your function will be called from the same thread that instantiated it.
- The same instance of your ScalarFunction subclass can be called on to process multiple blocks of data.
- The rows of input sent to processBlock() are not guaranteed to be any particular order.
- Writing too many output rows can cause Vertica to emit an out-of-bounds error.
ScalarFunctionFactory

The ScalarFunctionFactory class tells Vertica metadata about your UDSF: its number of parameters and their data types, as well as the data type of its return value. It also instantiates a subclass of ScalarFunction.

Methods

You must implement the following methods in your ScalarFunctionFactory subclass:

- `createScalarFunction()` instantiates a ScalarFunction subclass. If writing in C++, you can call the `vt_createFuncObj` macro with the name of the ScalarFunction subclass. This macro takes care of allocating and instantiating the class for you.

- `getPrototype()` tells Vertica about the parameters and return type for your UDSF. In addition to a ServerInterface object, this method gets two `ColumnType` objects. All you need to do in this function is to call class functions on these two objects to build the list of parameters and the single return value type.

After defining your factory class, you need to call the `RegisterFactory` macro. This macro instantiates a member of your factory class, so Vertica can interact with it and extract the metadata it contains about your UDSF.

Declaring Return Values

If your function returns a sized column (a return data type whose length can vary, such as a VARCHAR) or a value that requires precision, you must implement `getReturnType()`. This method is called by Vertica to find the length or precision of the data being returned in each row of the results. The return value of this method depends on the data type your `processBlock()` method returns:

- CHAR or VARCHAR return the maximum length of the string.
- NUMERIC types specify the precision and scale.
- TIME and TIMESTAMP values (with or without timezone) specify precision.
- INTERVAL YEAR TO MONTH specifies range.
- INTERVAL DAY TO SECOND specifies precision and range.
If your UDSF does not return one of these data types, it does not need a `getReturnType()` method.

The input to the `getReturnType()` method is a `SizedColumnTypes` object that contains the input argument types along with their lengths. This object will be passed to an instance of your `processBlock()` function. Your implementation of `getReturnType()` must extract the data types and lengths from this input and determine the length or precision of the output rows. It then saves this information in another instance of the `SizedColumnTypes` class.

### Setting Null Input and Volatility Behavior

Normally, Vertica calls your UDSF for every row of data in the query. There are some cases where Vertica can avoid executing your UDSF. You can tell Vertica when it can skip calling your function and just supply a return value itself by changing your function's volatility and strictness settings.

- **Your function's volatility** indicates whether it always returns the same output value when passed the same arguments. Depending on its behavior, Vertica can cache the arguments and the return value. If the user calls the UDSF with the same set of arguments, Vertica returns the cached value instead of calling your UDSF.

- **Your function's strictness** indicates how it reacts to NULL arguments. If it always returns NULL when *any* argument is NULL, Vertica can just return NULL without having to call the function. This optimization also saves you work, because you do not need to test for and handle null arguments in your UDSF code.

You indicate the volatility and null handling of your function by setting the `vol` and `strict` fields in your `ScalarFunctionFactory` class's constructor.

### Volatility Settings

To indicate your function's volatility, set the `vol` field to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLATILE</td>
<td>Repeated calls to the function with the same arguments always result in different values. Vertica always calls volatile functions for each invocation.</td>
</tr>
<tr>
<td>IMMUTABLE</td>
<td>Calls to the function with the same arguments always results in the same return value.</td>
</tr>
<tr>
<td>STABLE</td>
<td>Repeated calls to the function with the same arguments <em>within the same</em></td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>statement</td>
<td>returns the same output. For example, a function that returns the current user name is stable because the user cannot change within a statement. The user name could change between statements.</td>
</tr>
<tr>
<td>DEFAULT_VOLATILITY</td>
<td>The default volatility. This is the same as VOLATILE.</td>
</tr>
</tbody>
</table>

**C++ Example**

The following example shows a version of the Add2ints example factory class that makes the function immutable.

```cpp
class Add2intsImmutableFactory : public Vertica::ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    {
        return vt_createFuncObj(srvInterface.allocator, Add2ints); }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
                              Vertica::ColumnTypes &argTypes,
                              Vertica::ColumnTypes &returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    public:
        Add2intsImmutableFactory() {vol = IMMUTABLE;}
};
RegisterFactory(Add2intsImmutableFactory);
```

**Java Example**

The following example demonstrates setting the Add2IntsFactory's vol field to IMMUTABLE to tell Vertica it can cache the arguments and return value.

```java
public class Add2IntsFactory extends ScalarFunctionFactory {

    @Override
    public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType){
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface){
        return new Add2Ints();
    }
```
Null Input Behavior

To indicate how your function reacts to NULL input, set the strictness field to one of the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLED_ON_NULL_INPUT</td>
<td>The function must be called, even if one or more arguments are NULL.</td>
</tr>
<tr>
<td>RETURN_NULL_ON_NULL_INPUT</td>
<td>The function always returns a NULL value if any of its arguments are NULL.</td>
</tr>
<tr>
<td>STRICT</td>
<td>A synonym for RETURN_NULL_ON_NULL_INPUT</td>
</tr>
<tr>
<td>DEFAULT_STRICTNESS</td>
<td>The default strictness setting. This is the same as CALLED_ON_NULL_INPUT.</td>
</tr>
</tbody>
</table>

C++ Example

The following example demonstrates setting the null behavior of Add2ints so Vertica does not call the function with NULL values.

class Add2intsNullOnNullInputFactory : public Vertica::ScalarFunctionFactory
{
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    { return vt_createFuncObj(srvInterface.allocate, Add2ints); }
    virtual void getPrototype(Vertica::ServerInterface &srvInterface,
                               Vertica::ColumnType &argTypes,
                               Vertica::ColumnType &returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }
}

public:
Add2intsNullOnNullInputFactory() {strict = RETURN_NULL_ON_NULL_INPUT;}
The following example demonstrates setting the strictness setting of the Add2Ints example to STRICT. This means that if either of the two values to be added is NULL, Vertica can set the return value to NULL without having to call the Add2Ints function.

```java
public class Add2IntsFactory extends ScalarFunctionFactory {
    @Override
    public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType){
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    @Override
    public ScalarFunction createScalarFunction(ServerInterface srvInterface){
        return new Add2Ints();
    }

    public Add2IntsFactory() {
        // Tell Vertica that any NULL arguments results in a NULL return value
        strict = strictness.RETURN_NULL_ON_NULL_INPUT;
    }
}
```

### C++ API

This section provides APIs and examples for the C++ API for UDSFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

### ScalarFunction and ScalarFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See the [UDSF Class Overview](#) for general information about implementing the ScalarFunction and ScalarFunctionFactory classes.

### ScalarFunction API

The API provides the following methods for extension by subclasses:
virtual void setup(ServerInterface &srvInterface,
    const SizedColumnTypes &argTypes);

virtual void processBlock(ServerInterface &srvInterface,
    BlockReader &arg_reader, BlockWriter &res_writer)=0;

virtual void destroy(ServerInterface &srvInterface, const SizedColumnTypes &argTypes);

ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

virtual ScalarFunction * createScalarFunction(ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
    ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getReturnType(ServerInterface &srvInterface,
    const SizedColumnTypes &argTypes, SizedColumnTypes &returnType);

virtual void getParameterType(ServerInterface &srvInterface,
    SizedColumnTypes &parameterTypes);

C++ Example: Add2Ints

The following example shows a very basic subclass of ScalarFunction called Add2Ints. As the name implies it adds two integers together, returning a single integer result. It also demonstrates including the main Vertica SDK header file (Vertica.h) and using the Vertica namespace. While not required, using the namespace saves you from having to prefix every Vertica SDK class reference with Vertica::.

ScalarFunction Implementation

// Include the top-level Vertica SDK file
#include "Vertica.h"
// Using the Vertica namespace means we don’t have to prefix all
// class references with Vertica::
using namespace Vertica;
/*
 * ScalarFunction implementation for a UDSF that adds
 * two numbers together.
 */
class Add2Ints : public ScalarFunction
{
    public:
    /*
* This function does all of the actual processing for the UDF.
* In this case, it simply reads two integer values and returns
* their sum.
* The inputs are retrieved via arg_reader
* The outputs are returned via arg_writer
*/
virtual void processBlock(ServerInterface &srvInterface,
   BlockReader &arg_reader,
   BlockWriter &res_writer)
{
   // While we have input to process
   do {
      // Read the two integer input parameters by calling the
      // BlockReader.getIntRef class function
      const vint a = arg_reader.getIntRef(0);
      const vint b = arg_reader.getIntRef(1);
      // Call BlockWriter.setInt to store the output value, which is the
      // two input values added together
      res_writer.setInt(a+b);
      // Finish writing the row, and advance to the next output row
      res_writer.next();
      // Continue looping until there are no more input rows
   } while (arg_reader.next());
};

ScalarFunctionFactory Implementation

/*
* This class provides metadata about the ScalarFunction class, and
* also instantiates a member of that class when needed.
*/
class Add2IntsFactory : public ScalarFunctionFactory
{
   // return an instance of Add2Ints to perform the actual addition.
   virtual ScalarFunction *createScalarFunction(ServerInterface &interface)
   {
      // Calls the vt_createFuncObj to create the new Add2Ints class instance.
      return vt_createFuncObj(interface.allocator, Add2Ints);
   }
   // This function returns the description of the input and outputs of the
   // Add2Ints class's processBlock function. It stores this information in
   // two ColumnTypes objects, one for the input parameters, and one for
   // the return value.
   virtual void getPrototype(ServerInterface &interface,
      ColumnTypes &argTypes,
      ColumnTypes &returnType)
   {
      // Takes two ints as inputs, so add ints to the argTypes object
      argTypes.addInt();
      argTypes.addInt();
      // returns a single int, so add a single int to the returnType object.
      // Note that ScalarFunctions *always* return a single value.
      returnType.addInt();
   }
};
The RegisterFactory Macro

Use the RegisterFactory macro to register a ScalarFunctionFactory subclass. This macro instantiates the factory class and makes the metadata it contains available for Vertica to access. To call this macro, pass it the name of your factory class.

```c++
RegisterFactory(Add2IntsFactory);
```

C++ Example: Calling a UDSF from a Check Constraint

This example shows you the C++ code needed to create a UDSF that can be called by a check constraint. The name of the sample function is LargestSquareBelow. The sample function determines the largest number whose square is less than the number in the subject column. For example, if the number in the column is 1000, the largest number whose square is less than 1000 is 31 (961).

Important: A UDSF used within a check constraint must be immutable, and the constraint must handle null values properly. Otherwise, the check constraint might not work as you intended. In addition, Vertica evaluates the predicate of an enabled check constraint on every row that is loaded or updated, so consider performance in writing your function.

For information on check constraints, see Check Constraints the Administrator's Guide.
For information about implementing the ScalarFunction and ScalarFunctionFactory classes, see UDSF Class Overview.

Loading and Using the Example

The following example shows how you can create and load a library named MySqLib, using CREATE LIBRARY. Adjust the library path in this example to the absolute path and file name for the location where you saved the shared object LargestSquareBelow.

```sql
=> CREATE OR REPLACE LIBRARY MySqLib AS '/home/dbadmin/LargestSquareBelow.so';
```

1. After you create and load the library, add the function to the catalog using the CREATE FUNCTION (UDF) statement.
CREATE OR REPLACE FUNCTION largestSqBelow AS LANGUAGE 'C++' NAME 'LargestSquareBelowInfo' LIBRARY MySqLib;

2. Next, include the UDSF in a check constraint.

CREATE TABLE squaretest(
    ceiling INTEGER UNIQUE,
    CONSTRAINT chk_sq CHECK (largestSqBelow(ceiling) < ceiling*ceiling)
);

3. Add data to the table, squaretest.

COPY squaretest FROM stdin DELIMITER ',null';
-1
null
0
1
1000
1000000
1000001

Your output should be similar to the following sample, based upon the data you use:

SELECT ceiling, largestSqBelow(ceiling)
FROM squaretest ORDER BY ceiling;

<table>
<thead>
<tr>
<th>ceiling</th>
<th>largestSqBelow</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>31</td>
</tr>
<tr>
<td>1000000</td>
<td>999</td>
</tr>
<tr>
<td>1000001</td>
<td>1000</td>
</tr>
</tbody>
</table>
(7 rows)

ScalarFunction Implementation

This ScalarFunction implementation does the processing work for a UDSF that determines the largest number whose square is less than the number input.

#include "Vertica.h"
/*
 * ScalarFunction implementation for a UDSF that
 * determines the largest number whose square is less than
 * the number input.
 */
class LargestSquareBelow : public Vertica::ScalarFunction {

public:
/*
 This function does all of the actual processing for the UDSF.
* The inputs are retrieved via arg_reader
* The outputs are returned via arg_writer
*/
virtual void processBlock(Vertica::ServerInterface &srvInterface,
Vertica::BlockReader &arg_reader,
Vertica::BlockWriter &res_writer)
{
    if (arg_reader.getNumCols() != 1)
        vt_report_error(0, "Function only accept 1 argument, but %zu provided", arg_reader.getNumCols());
    // While we have input to process
    do {
        // Read the input parameter by calling the BlockReader.getIntRef class function
        const Vertica::vint a = arg_reader.getIntRef(0);
        Vertica::vint res;
        // Determine the largest square below the number
        if ((a != Vertica::vint_null) && (a > 0))
            res = (Vertica::vint)sqrt(a - 1);
        else
            res = Vertica::vint_null;
        // Call BlockWriter.setInt to store the output value,
        // which is the largest square
        res_writer.setInt(res);
        // Write the row and advance to the next output row
        res_writer.next();
        // Continue looping until there are no more input rows
    } while (arg_reader.next());
};

ScalarFunctionFactory Implementation
This ScalarFunctionFactory implementation does the work of handling input and output, and marks the function as immutable (a requirement if you plan to use the UDSF within a check constraint).

class LargestSquareBelowInfo : public Vertica::ScalarFunctionFactory
{
    //return an instance of LargestSquareBelow to perform the computation.
    virtual Vertica::ScalarFunction *createScalarFunction(Vertica::ServerInterface &srvInterface)
    //Call the vt_createFuncObj to create the new LargestSquareBelow class instance.
    { return Vertica::vt_createFuncObject<LargestSquareBelow>(srvInterface.allocator); }

    /*
    * This function returns the description of the input and outputs of the
    * LargestSquareBelow class’s processBlock function. It stores this information in
    * two ColumnTypes objects, one for the input parameter, and one for
    * the return value.
/*
   virtual void getPrototype(Vertica::ServerInterface &srvInterface,
       Vertica::ColumnTypes &argTypes,
       Vertica::ColumnTypes &returnType)
   {
       // Takes one int as input, so adds int to the argTypes object
       argTypes.addInt();
       // Returns a single int, so add a single int to the returnType object.
       // ScalarFunctions always return a single value.
       returnType.addInt();
   }
public:
   // the function cannot be called within a check constraint unless the UDx author
   // certifies that the function is immutable:
   LargestSquareBelowInfo() { vol = Vertica::IMMUTABLE; }
};

The RegisterFactory Macro

Use the RegisterFactory macro to register a ScalarFunctionFactory subclass. This
macro instantiates the factory class and makes the metadata it contains available for Vertica to
access. To call this macro, pass it the name of your factory class.

RegisterFactory(LargestSquareBelowInfo);

Java API

This section provides APIs and examples for the Java API for UDSFs.
For information on setting up a Java development environment and compiling and packaging
libraries, see Developing with the Java SDK.

ScalarFunction and ScalarFunctionFactory Java Interface

This section describes information that is specific to the Java API. See UDSF Class Overview for
general information about implementing the ScalarFunction and
ScalarFunctionFactory classes.

ScalarFunction API

The API provides the following methods for extension by subclasses:

void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);
abstract void processBlock(ServerInterface srvInterface, BlockReader arg_reader, BlockWriter res_writer) throws UdfException, DestroyInvocation;
void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);

ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

abstract ScalarFunction createScalarFunction(ServerInterface srvInterface);
abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);
void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes, SizedColumnTypes returnType) throws UdfException;
void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);

Java Example: Add2Ints

The Add2Ints scalar function adds two integers together, returning a single integer result.

Loading and Using the Example

Use CREATE LIBRARY to load the jar file containing the function, and then use CREATE FUNCTION (UDF) to declare the function as in the following example:

=> CREATE FUNCTION add2ints AS LANGUAGE 'Java' NAME 'com.mycompany.example.Add2intsFactory' LIBRARY add2intslib;

The following example shows how to use this function.

=> SELECT Add2Ints(27,15);
Add2int
----------
42
(1 row)
=> SELECT * FROM MyTable;
a | b
----------
7 | 0
12 | 2
12 | 6
18 | 9
1 | 1
Implementation

The following code example is the full source of the Java Add2ints example. To simplify handling the source code, the ScalarFunction class is defined as an inner class of the ScalarFunctionFactory class.

```java
// You will need to specify the full package when creating functions based on
// the classes in your library.
package com.mycompany.example;
// Import all of the Vertica SDK
import com.vertica.sdk.*;
public class Add2intsFactory extends ScalarFunctionFactory
{
    @Override
    public void getPrototype(ServerInterface srvInterface,
        ColumnTypes argTypes,
        ColumnTypes returnType)
    {
        argTypes.addInt();
        argTypes.addInt();
        returnType.addInt();
    }

    // This ScalarFunction is defined as an inner class of
    // its ScalarFunctionFactory class. This gets around having
    // to have a separate source file for this public class.
    public class Add2ints extends ScalarFunction
    {
        @Override
        public void processBlock(ServerInterface srvInterface,
            BlockReader argReader,
            BlockWriter resWriter)
            throws UdfException, DestroyInvocation
        {
            do {
                // The input and output objects have already loaded
                // the first row, so you can start reading and writing
                // values immediately.

                // Get the two integer arguments from the BlockReader
                long a = argReader.getLong(0);
                long b = argReader.getLong(1);

                // Process the arguments and come up with a result. For this
```
// example, just add the two arguments together.
long result = a+b;

// Write the integer output value.
resWriter.setLong(result);

// Advance the output BlockWriter to the next row.
resWriter.next();

// Continue processing input rows until there are no more.
} while (argReader.next());

@Override
public ScalarFunction createScalarFunction(ServerInterface srvInterface) {
    return new Add2ints();
}
ScalarFunctionFactory API

The API provides the following methods for extension by subclasses:

class python_udx_factory(vertica_sdk.ScalarFunctionFactory):
    def createScalarFunction(self, srv):
        pass
    def getPrototype(self, srv_interface, arg_types, return_type):
        pass
    def getReturnType(self, srv_interface, arg_types, return_type):
        pass

Python Example: add2ints

The `add2ints` scalar function adds two integers together, returning a single integer result.

You can find more UDx examples in the Vertica Github repository, https://github.com/vertica/UDx-Examples.

UDSF Python Code

```python
import vertica_sdk

class add2ints(vertica_sdk.ScalarFunction):
    """Return the sum of two integer columns""
    def __init__(self):
        pass
    def setup(self, server_interface, col_types):
        pass
    def processBlock(self, self, server_interface, arg_reader, res_writer):
        # Writes a string to the UDx log file.
        server_interface.log("Python UDx - Adding 2 ints!")
        while(True):
            # Example of error checking best practices.
            if arg_reader.isNull(0) or arg_reader.isNull(1):
                raise ValueError("I found a NULL!")
            else:
                first_int = arg_reader.getInt(0) # Input column
                second_int = arg_reader.getInt(1) # Input column
                res_writer.setInt(first_int + second_int) # Sum of input columns.
                res_writer.next() # Read the next row.
                if not arg_reader.next():
                    # Stop processing when there are no more input rows.
                    break
    def destroy(self, self, server_interface, col_types):
```

Vertica Analytic Database (9.0.x)
pass

class add2ints_factory(vertica_sdk.ScalarFunctionFactory):
    
def createScalarFunction(self, srv):
        return add2ints()
    
def getPrototype(self, srv_interface, arg_types, return_type):
        arg_types.addInt()
        arg_types.addInt()
        return_type.addInt()
    
def getReturnType(self, srv_interface, arg_types, return_type):
        return_type.addInt()

Load the Function and Library

Create the library and the function.

=> CREATE LIBRARY pylib AS '/home/dbadmin/python_udx/add2ints/add2ints.py' LANGUAGE 'Python';
CREATE LIBRARY
=> CREATE FUNCTION add2ints AS LANGUAGE 'Python' NAME 'add2ints_factory' LIBRARY pylib fenced;
CREATE FUNCTION

Querying Data with the Function

The following shows how to run a query with the UDSF.

=> SELECT add2ints(10, 10);
add2ints
----------
  20
(1 row)

You can query on a table with two integer columns as follows:

=> SELECT nums_1, nums_2, add2ints(nums_1, nums_2) AS add2ints_sum
FROM bunch_of_numbers;
nums_1 | nums_2 | add2ints_sum |
----------|---------|---------------
  10 |  10 | 20 |
  1 |  4 |  5 |
  6 |  6 | 12 |
 30 | 144 | 174 |
(4 rows)
Python Example: currency_convert

The currency_convert scalar function reads two values from a table, a currency and a value. It then converts the item’s value to USD, returning a single float result.

You can find more UDX examples in the Vertica Github repository, https://github.com/vertica/UDx-Examples.

UDSF Python Code

```python
import vertica_sdk
import decimal

rates2USD = {'USD': 1.000,
             'EUR': 0.89977,
             'GBP': 0.68452,
             'INR': 67.0345,
             'AUD': 1.39187,
             'CAD': 1.30335,
             'ZAR': 15.7181,
             'XXX': -1.0000}

class currency_convert(vertica_sdk.ScalarFunction):
    """Converts a money column to another currency

    Returns a value in USD.
    """
    def __init__(self):
        pass

def setup(self, server_interface, col_types):
    pass

def processBlock(self, server_interface, block_reader, block_writer):
    while(True):
        currency = block_reader.getString(0)
        try:
            rate = decimal.Decimal(rates2USD[currency])
        except KeyError:
            server_interface.log("ERROR: {} not in dictionary.".format(currency))
        # Scalar functions always need a value to move forward to the
        # next input row. Therefore, we need to assign it a value to
        # move beyond the error.
        currency = 'XXX'
        rate = decimal.Decimal(rates2USD[currency])

        starting_value = block_reader.getNumeric(1)
        converted_value = decimal.Decimal(starting_value) / rate
        block_writer.setNumeric(converted_value)
        block_writer.next()
        if not block_reader.next():
```
break

def destroy(self, server_interface, col_types):
    pass

class currency_convert_factory(vertica_sdk.ScalarFunctionFactory):
    
def createScalarFunction(self, srv):
        return currency_convert()

    def getPrototype(self, srv_interface, arg_types, return_type):
        arg_types.addVarchar()
        arg_types.addNumeric()
        return_type.addNumeric()

    def getReturnType(self, srv_interface, arg_types, return_type):
        return_type.addNumeric(9,4)

Load the Function and Library

Create the library and the function.

=> CREATE LIBRARY pylib AS '/home/dbadmin/python_udx/currency_convert/currency_convert.py' LANGUAGE 'Python';
CREATE LIBRARY
=> CREATE FUNCTION currency_convert AS LANGUAGE 'Python' NAME 'currency_convert_factory' LIBRARY pylib fenced;
CREATE FUNCTION

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

=> SELECT product, currency_convert(currency, value) AS cost_in_usd
    FROM items;

<table>
<thead>
<tr>
<th>product</th>
<th>cost_in_usd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoes</td>
<td>133.4008</td>
</tr>
<tr>
<td>Soccer Ball</td>
<td>110.2817</td>
</tr>
<tr>
<td>Coffee</td>
<td>13.5190</td>
</tr>
<tr>
<td>Surfboard</td>
<td>176.2593</td>
</tr>
<tr>
<td>Hockey Stick</td>
<td>76.7177</td>
</tr>
<tr>
<td>Car</td>
<td>17000.0000</td>
</tr>
<tr>
<td>Software</td>
<td>10.4424</td>
</tr>
<tr>
<td>Hamburger</td>
<td>7.5000</td>
</tr>
<tr>
<td>Fish</td>
<td>130.4272</td>
</tr>
<tr>
<td>Cattle</td>
<td>269.2367</td>
</tr>
</tbody>
</table>

(10 rows)
Python Example: validate_url

The `validate_url` scalar function reads a string from a table, a URL. It then validates if the URL is responsive, returning a status code or a string indicating the attempt failed.

You can find more UDx examples in the Vertica Github repository, https://github.com/vertica/UDx-Examples.

UDSF Python Code

```python
import vertica_sdk
import urllib.request
import time

class validate_url(vertica_sdk.ScalarFunction):
    """Validates HTTP requests.

    Returns the status code of a webpage. Pages that cannot be accessed return "Failed to load page."
    ""

    def __init__(self):
        pass

    def setup(self, server_interface, col_types):
        pass

    def processBlock(self, server_interface, arg_reader, res_writer):
        # Writes a string to the UDx log file.
        server_interface.log("Validating webpage accessibility - UDx")
        while(True):
            url = arg_reader.getString(0)
            try:
                status = urllib.request.urlopen(url).getcode()
                # Avoid overwhelming web servers -- be nice.
                time.sleep(2)
            except (ValueError, urllib.error.HTTPError, urllib.error.URLError):
                status = 'Failed to load page'
            res_writer.setString(str(status))
            res_writer.next()  # Stop processing when there are no more input rows.
            break

    def destroy(self, server_interface, col_types):
        pass

class validate_url_factory(vertica_sdk.ScalarFunctionFactory):
    def createScalarFunction(self, srv):
        return validate_url()
```
def getPrototype(self, srv_interface, arg_types, return_type):
    arg_types.addVarchar()
    return_type.addChar()

def getReturnType(self, srv_interface, arg_types, return_type):
    return_type.addChar(20)

Load the Function and Library

Create the library and the function.

=> CREATE OR REPLACE LIBRARY pylib AS 'webpage_tester/validate_url.py' LANGUAGE 'Python';
=> CREATE OR REPLACE FUNCTION validate_url AS LANGUAGE 'Python' NAME 'validate_url_factory' LIBRARY pylib fenced;

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

=> SELECT url, validate_url(url) AS url_status FROM webpages;

<table>
<thead>
<tr>
<th>url</th>
<th>url_status</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.vertica.com/documentation/vertica/">http://www.vertica.com/documentation/vertica/</a></td>
<td>200</td>
</tr>
<tr>
<td><a href="http://www.google.com/">http://www.google.com/</a></td>
<td>200</td>
</tr>
<tr>
<td><a href="http://www.mass.gov.com/">http://www.mass.gov.com/</a></td>
<td>Failed to load page</td>
</tr>
<tr>
<td><a href="http://www.espn.com">http://www.espn.com</a></td>
<td>200</td>
</tr>
<tr>
<td><a href="http://blah.blah.blah.blah">http://blah.blah.blah.blah</a></td>
<td>Failed to load page</td>
</tr>
<tr>
<td><a href="http://www.vertica.com/">http://www.vertica.com/</a></td>
<td>200</td>
</tr>
</tbody>
</table>

(6 rows)

R API

This section provides APIs and examples for the R API for UDSFs.

For information on setting up an R development environment and compiling and packaging libraries, see Developing with the R SDK.

Scalar Function and Scalar Function Factory R Interface

This section describes information that is specific to the R API. See R SDK API Documentation for general information about implementing Scalar functions.
Scalar Function API

The API provides the following framework for extension by R function:

```r
FunctionName <- function(input.data.frame, parameters.data.frame) {
    # Computations
    # The function must return a data frame.
    return(output.data.frame)
}
```

Scalar Function Factory API

The API provides the following framework for extension by R function:

```r
FunctionNameFactory <- function() {
    list(name = FunctionName,
         udxtype = c("scalar"),
         intype = c("int"),
         outtype = c("int"))
}
```

For a complete list of factory options, see the R API documentation for Factory Function.

R Example: SalesTaxCalculator

The SalesTaxCalculator scalar function reads a float and a varchar from a table, an item's price and the state abbreviation. It then uses the state abbreviation to find the sales tax rate from a list and calculates the item's price including the state's sales tax, returning the total cost of the item.

You can find more UDx examples in the Vertica Github repository, https://github.com/vertica/UDx-Examples.

Load the Function and Library

Create the library and the function.

```sql
=> CREATE OR REPLACE LIBRARY rLib AS 'sales_tax_calculator.R' LANGUAGE 'R';
CREATE LIBRARY
=> CREATE OR REPLACE FUNCTION SalesTaxCalculator AS LANGUAGE 'R' NAME 'SalesTaxCalculatorFactory'
LIBRARY rLib FENCED;
CREATE FUNCTION
```
Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```
=> SELECT item, state_abbreviation, price, SalesTaxCalculator(price, state_abbreviation) AS Price_With_Sales_Tax
    FROM inventory;
```

<table>
<thead>
<tr>
<th>item</th>
<th>state_abbreviation</th>
<th>price</th>
<th>Price_With_Sales_Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scarf</td>
<td>AZ</td>
<td>6.88</td>
<td>7.53016</td>
</tr>
<tr>
<td>Software</td>
<td>MA</td>
<td>88.31</td>
<td>96.655295</td>
</tr>
<tr>
<td>Soccer Ball</td>
<td>MS</td>
<td>12.55</td>
<td>13.735975</td>
</tr>
<tr>
<td>Beads</td>
<td>LA</td>
<td>0.99</td>
<td>1.083555</td>
</tr>
<tr>
<td>Baseball</td>
<td>TN</td>
<td>42.42</td>
<td>46.42869</td>
</tr>
<tr>
<td>Cheese</td>
<td>WI</td>
<td>20.77</td>
<td>22.732765</td>
</tr>
<tr>
<td>Coffee Mug</td>
<td>MA</td>
<td>8.99</td>
<td>9.839555</td>
</tr>
<tr>
<td>Shoes</td>
<td>TN</td>
<td>23.99</td>
<td>26.257055</td>
</tr>
</tbody>
</table>

(8 rows)

UDSF R Code

```R
SalesTaxCalculator <- function(input.data.frame) {
  # Not a complete list of states in the USA, but enough to get the idea.
  state.sales.tax <- list(ma = 0.0625,
                          az = 0.087,
                          la = 0.0891,
                          tn = 0.0945,
                          wi = 0.0543,
                          ms = 0.0707)
  for (state_abbreviation in input.data.frame[, 2]) {
    # Ensure state abbreviations are lowercase.
    lower_state <- tolower(state_abbreviation)
    # Check if the state is in our state.sales.tax list.
    if (is.null(state.sales.tax[[lower_state]])) {
      stop("State is not in our small sample!")
    } else {
      sales.tax.rate <- state.sales.tax[[lower_state]]
      item.price <- input.data.frame[, 1]
      # Calculate the price including sales tax.
      price.with.sales.tax <- (item.price) + (item.price * sales.tax.rate)
    }
  }
  return(price.with.sales.tax)
}

SalesTaxCalculatorFactory <- function() {
  list(name = SalesTaxCalculator,
       udxtype = c("scalar"),
       intype = c("float", "varchar"),
       outtype = c("float"))
}
```
R Example: kmeans

The KMeans_User scalar function reads any number of columns from a table, the observations. It then uses the observations and the two parameters when applying the kmeans clustering algorithm to the data, returning an integer value associated with the cluster of the row.

You can find more UDx examples in the VerticaGithub repository, https://github.com/vertica/UDx-Examples.

Load the Function and Library

Create the library and the function.

```sql
=> CREATE OR REPLACE LIBRARY rLib AS 'kmeans.R' LANGUAGE 'R';
CREATE LIBRARY
=> CREATE OR REPLACE FUNCTION KMeans_User AS LANGUAGE 'R' NAME 'KMeans_UserFactory' LIBRARY rLib FENCED;
CREATE FUNCTION
```

Querying Data with the Function

The following query shows how you can run a query with the UDSF.

```sql
=> SELECT spec, KMeans_User(s1, sw, pl, pw USING PARAMETERS clusters = 3, nstart = 20)  
    FROM iris;
```

<table>
<thead>
<tr>
<th>spec</th>
<th>KMeans_User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td>2</td>
</tr>
</tbody>
</table>

(150 rows)
UDSF R Code

```r
KMeans_User <- function(input.data.frame, parameters.data.frame) {
  # Take the clusters and nstart parameters passed by the user and assign them
  # to variables in the function.
  if (is.null(parameters.data.frame[['clusters']])) {
    stop("NULL value for clusters! clusters cannot be NULL.")
  } else {
    clusters.value <- parameters.data.frame[['clusters']]}
  if (is.null(parameters.data.frame[['nstart']])) {
    stop("NULL value for nstart! nstart cannot be NULL.")
  } else {
    nstart.value <- parameters.data.frame[['nstart']]}
  # Apply the algorithm to the data.
  kmeans.clusters <- kmeans(input.data.frame[, 1:length(input.data.frame)],
    clusters.value, nstart = nstart.value)
  final.output <- data.frame(kmeans.clusters$cluster)
  return(final.output)
}
KMeans_UserFactory <- function() {
  list(name = KMeans_User,
    ud xtype = c("scalar"),
    # Since this is a polymorphic function the intype must be any
    intype = c("any"),
    outtype = c("int"),
    parametertypecallback=KMeansParameters)
}
KMeansParameters <- function() {
  parameters <- list(datatype = c("int", "int"),
    length = c("NA", "NA"),
    scale = c("NA", "NA"),
    name = c("clusters", "nstart"))
  return(parameters)
}
```

Vertica Documentation
Extending Vertica
Transform Functions (UDTFs)

A User-Defined Transform Function (UDTF) lets you transform a table of data into another table. It reads one or more arguments (treated as a row of data), and returns zero or more rows of data consisting of one or more columns. A UDTF can produce any number of rows as output. However, each row it outputs must be complete. Advancing to the next row without having added a value for each column produces incorrect results.

The schema of the output table does not need to correspond to the schema of the input table—they can be totally different. The UDTF can return any number of output rows for each row of input.

UDTFs can only be used in the SELECT list that contains just the UDTF call and a required OVER clause. A multi-phase UDTF can make use of partition columns (PARTITION BY), but other UDTFs cannot.

UDTFs are run after GROUP BY, but before the final ORDER BY, when used in conjunction with GROUP BY and ORDER BY in a statement. The ORDER BY clause may contain only columns or expressions that are in a window partition clause (see Window Partitioning).

UDTFs can take up to 1600 parameters (input columns). Attempting to pass more parameters to a UDTF results in an error.

UDTF Class Overview

You create your UDTF by subclassing two classes defined by the Vertica SDK: TransformFunction and TransformFunctionFactory.

The TransformFunctionFactory performs two roles:

- It provides the number of parameters and their data types accepted by the UDTF and the number of output columns and their data types UDTF's output. Vertica uses this data when you call the CREATE TRANSFORM FUNCTION SQL statement to add the function to the database catalog.

- It returns an instance of the UDTF function's TransformFunction subclass that Vertica can call to process data.
TransformFunction

The TransformFunction class is where you perform the data-processing, transforming input rows into output rows. Your subclass must define the processPartition() method. It may define methods to set up and tear down the function.

Performing the Transformation

The processPartition() method carries out all of the processing that you want your UDTF to perform. When a user calls your function in a SQL statement, Vertica bundles together the data from the function parameters and passes it to processPartition().

The input and output of the processPartition() method are supplied by objects of the BlockReader and BlockWriter classes. They define methods that you use to read the input data and write the output data for your UDTF.

Your processPartition() method should follow this basic pattern:

- Extract the input parameters by calling a data-type-specific function in the PartitionReader object to extract each input parameter. All of these functions take a single parameter: the column number in the input row that you want to read. Your function might need to handle NULL values.

- Perform the actual transformation.

- Write the output, if any. Unlike a UDSF, outputting data is optional for a UDTF. However, if it does write output, it must supply values for all of the output columns you defined in your factory. Similarly to reading input columns, the PartitionWriter object has functions for writing each type of data to the output row.

- Advance to the next row by calling ProcessReader.next(). This function returns true if there is another row of input data to process and false if all the data in the partition has been read. Once the input rows are exhausted, your UDTF exits so its results are returned back to Vertica.

Note: In some cases, you may want to determine the number and types of parameters using PartitionReader's getNumCols() and getTypeMetaData() functions, instead of just hard-coding the data types of the columns in the input row. This is useful if you want your TransformFunction to be able to process input tables with different schemas. You can then use different TransformFunctionFactory classes to define multiple function signatures that call the same TransformFunction class. See Handling.
**Different Numbers and Types of Arguments** for more information.

Setting Up and Tearing Down

The `TransformFunction` class defines two additional methods that you can optionally implement to allocate and free resources: `setup()` and `destroy()`. You should use these methods to allocate and deallocate resources that you do not allocate through the UDx API (see *Allocating Resources for UDxs* for details).

**TransformFunctionFactory**

The `TransformFunctionFactory` class tells Vertica metadata about your UDTF: its number of parameters and their data types, as well as the data type of its return value. It also instantiates a subclass of `TransformFunction`.

You must implement the following methods in your `TransformFunctionFactory`:

- `getPrototype()` returns two `ColumnType` objects that describe the columns your UDTF takes as input and returns as output.
- `createTransformFunction()` instantiates your `TransformFunction` subclass.
- `getReturnType()` tells Vertica details about the output values: the width of variable-sized data types (such as `VARCHAR`) and the precision of data types that have settable precision (such as `TIMESTAMP`). You can also set the names of the output columns using this function.

**Note**: The `getReturnType()` function is required for UDTFs. It is optional for UDxs that return single values, such as User-Defined Scalar Functions.

See Also

*Creating Multi-Phase UDTFs*

Creating Multi-Phase UDTFs

Multi-phase UDTFs let you break your data processing into multiple steps. Using this feature, your UDTFs can perform processing in a way similar to Hadoop or other MapReduce frameworks. You can use the first phase to break down and gather data, and then use subsequent phases to process the data. For example, the first phase of your UDTF could extract
specific types of user interactions from a web server log stored in the column of a table, and subsequent phases could perform analysis on those interactions.

Multi-phase UDTFs also let you decide where processing should occur: locally on each node, or throughout the cluster. If your multi-phase UDT is like a MapReduce process, you want the first phase of your multi-phase UDTF to process data that is stored locally on the node where the instance of the UDTF is running. This prevents large segments of data from being copied around the Vertica cluster. Depending on the type of processing being performed in later phases, you may choose to have the data segmented and distributed across the Vertica cluster.

Each phase of the UDTF is the same as a traditional (single-phase) UDTF: it receives a table as input, and generates a table as output. The schema for each phase's output does not have to match its input, and each phase can output as many or as few rows as it wants.

You create a subclass of TransformFunction to define the processing performed by each stage. If you already have a TransformFunction from a single-phase UDTF that performs the processing you want a phase of your multi-phase UDTF to perform, you can easily adapt it to work within the multi-phase UDTF.

What makes a multi-phase UDTF different from a traditional UDTF is the factory class you use. You define a multi-phase UDTF using a subclass of MultiPhaseTransformFunctionFactory, rather than the TransformFunctionFactory. This special factory class acts as a container for all of the phases in your multi-step UDTF. It provides Vertica with the input and output requirements of the entire multi-phase UDTF (through the getPrototype() function), and a list of all the phases in the UDTF.

Within your subclass of the MultiPhaseTransformFunctionFactory class, you define one or more subclasses of TransformFunctionPhase. These classes fill the same role as the TransformFunctionFactory class for each phase in your multi-phase UDTF. They define the input and output of each phase and create instances of their associated TransformFunction classes to perform the processing for each phase of the UDTF. In addition to these subclasses, your MultiPhaseTransformFunctionFactory includes fields that provide a handle to an instance of each of the TransformFunctionPhase subclasses.

**Partitioning Options for Processing Local Data**

UDTFs typically process data that is partitioned in a specific way. For example, a UDTF that processes a web server log file to count the number of hits referred by each partner web site needs to have its input partitioned by a referrer column. Each instance of the UDTF sees the hits referred by a particular partner site so it can count them.

In cases like this, the window partitioning clause should use a PARTITION BY clause. Each node in the cluster partitions the data it stores, sends some of these partitions off to other
nodes, and then consolidates the partitions it receives from other nodes and runs an instance of the UDTF to process them.

In other cases, a UDTF might not need to partition input data in a particular way—for example, a UDTF that parses data out of an Apache log file. In this case, you can specify that each UDTF instance process only the data that is stored locally by the node on which it is running. By eliminating the overhead of partitioning data, processing can be much more efficient.

You can tell a UDTF to process only local data with one of the following window partitioning options:

- **PARTITION BEST**: For thread-safe UDTFs only, optimizes performance through multi-threaded queries across multiple nodes.
- **PARTITION NODES**: Optimizes performance of single-threaded queries across multiple nodes.

The query must specify a source table that is replicated across all nodes and contains a single row (similar to the **DUAL** table). For example, the following statements call a UDTF that parses locally-stored Apache log files:

```plaintext
=> CREATE TABLE rep (dummy INTEGER) UNSEGMENTED ALL NODES;
CREATE TABLE
=> INSERT INTO rep VALUES (1);
OUTPUT
--------
1
(1 row)
=> SELECT ParseLogFile('/data/apache/log*') OVER (PARTITION BEST) FROM rep;
```

### C++ API

This section provides APIs and examples for the C++ API for UDTFs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

### TransformFunction and TransformFunctionFactory C++ Interface

This section describes information that is specific to the C++ API. See [UDTF Class Overview](#) for general information about implementing the TransformFunction and TransformFunctionFactory classes.
TransformFunction API

The API provides the following methods for extension by subclasses:

```cpp
virtual void setup(ServerInterface &srvInterface,
const SizedColumnTypes &argTypes);

virtual void processPartition(ServerInterface &srvInterface,
PartitionReader &input_reader, PartitionWriter &output_writer)=0;

virtual void destroy(ServerInterface &srvInterface,
const SizedColumnTypes &argTypes);
```

TransformFunctionFactory API

The API provides the following methods for extension by subclasses:

```cpp
virtual TransformFunction *
createTransformFunction (ServerInterface &srvInterface)=0;

virtual void getPrototype(ServerInterface &srvInterface,
ColumnTypes &argTypes, ColumnTypes &returnType)=0;

virtual void getReturnType(ServerInterface &srvInterface,
const SizedColumnTypes &argTypes,
SizedColumnTypes &returnType)=0;

virtual void getParameterType(ServerInterface &srvInterface,
SizedColumnTypes &parameterTypes);
```

MultiPhaseTransformFunctionFactory API

The MultiPhaseTransformFunctionFactory class extends TransformFunctionFactory The API provides the following additional methods for extension by subclasses:

```cpp
virtual void getPhases(ServerInterface &srvInterface,
std::vector<TransformFunctionPhase * > &phases)=0;
```

If using this factory you must also extend TransformFunctionPhase. See the SDK reference documentation.
C++ Example: String Tokenizer

The following example shows a subclass of TransformFunction named StringTokenizer. It defines a UDTF that reads a table containing an INTEGER ID column and a VARCHAR column. It breaks the text in the VARCHAR column into tokens (individual words). It returns a table containing each token, the row it occurred in, and its position within the string.

Loading and Using the Example

The following example shows how to load the function into Vertica. It assumes that the TransformFunctions.so library that contains the function has been copied to the dbadmin user's home directory on the initiator node.

```
=> CREATE LIBRARY TransformFunctions AS
   '/home/dbadmin/TransformFunctions.so';
CREATE LIBRARY
=> CREATE TRANSFORM FUNCTION tokenize
   AS LANGUAGE 'C++' NAME 'TokenFactory' LIBRARY TransformFunctions;
CREATE TRANSFORM FUNCTION
```

You can then use it from SQL statements, for example:

```
=> CREATE TABLE T (url varchar(30), description varchar(2000));
CREATE TABLE
=> INSERT INTO T VALUES ('www.amazon.com','Online retail merchant and provider of cloud services');
OUTPUT
-------
   1
(1 row)
=> INSERT INTO T VALUES ('www.hp.com','Leading provider of computer hardware and imaging solutions');
OUTPUT
-------
   1
(1 row)
=> INSERT INTO T VALUES ('www.vertica.com','World''s fastest analytic database');
OUTPUT
-------
   1
(1 row)
=> COMMIT;
COMMIT

=> -- Invoke the UDT
=> SELECT url, tokenize(description) OVER (partition by url) FROM T;
   url | words
-----------------------
www.amazon.com | Online
www.amazon.com | retail
www.amazon.com | merchant
www.amazon.com | and
www.amazon.com | provider
```
Notice that the number of rows and columns in the result table are different than the input table. This is one of the strengths of a UDTF.

TransformFunction Implementation

The following code defines the StringTokenizer class.

```cpp
#include "Vertica.h"
#include <sstream>
// Use the Vertica namespace to make referring
// to SDK classes easier.
using namespace Vertica;
using namespace std;

// The primary class for the StringTokenizer UDTF. This does the actual work.
class StringTokenizer : public TransformFunction {
    // Called for each partition in the table.
    virtual void processPartition(ServerInterface &srvInterface,
        PartitionReader &inputReader,
        PartitionWriter &outputWriter) {
        // Use a top-level try to prevent exceptions from bubbling up.
        try {
            // Loop through the input rows
            do {
                // First check if the id is NULL. If so, skip the row without
                // writing any output.
                if (inputReader.isNull(0))
                    {  
                        srvInterface.log("Skipping row with NULL ID");
                        continue;
                    }
                // Get an input row containing an int and a varchar
                const int row = inputReader.getIntRef(0);
                const VString &sentence = inputReader.getStringRef(1);
                // If input string is NULL, then output NULL tokens and
                // positions.
                if (sentence.isNull())
```
The `processPartition()` function in this example follows a pattern that you will follow in your own UDTF: it loops over all rows in the table partition that Vertica sends it, processing each row. For UDTFs you do not have to actually process every row. You can exit your function without having read all of the input without any issues. You may choose to do this if your UDTF is performing some sort search or some other operation where it can determine that the rest of the input is unneeded.

In this example, `processPartition()` first extracts the ID number of the row. Then it extracts the `VString` containing the text from the `PartitionReader` object. The `VString` class represents a Vertica string value (VARCHAR or CHAR).

When developing UDTFs, you often need to handle NULL input values in a special way. This example has two different reactions to a NULL value:
If the ID column is NULL, then processPartition() skips the row and moves on to the next row. It writes a message to the Vertica log to indicate that it is skipping the row.

If the VARCHAR column is NULL, processPartition() outputs a single row that contains the row_id value and NULLs for both the token and token_position columns.

After handling any NULL values, the processPartition() function shown in the example moves on to performing the actual processing. It breaks the string into individual tokens and outputs a row for each token.

Similarly to reading input columns, the PartitionWriter object has functions for writing each type of data to the output row. In this case, the example:

- Calls the PartitionWriter object's getStringRef() function to allocate a new VString object to hold the token to output for the first column. It then copies the token's value into the VString object.
- Calls setInt() to output the row ID in the second column.
- Calls setInt() again to output the token's position within the input string.

After setting the three column values, the example calls PartitionWriter.next() to complete the output row.

### TransformFunctionFactory Implementation

The following code shows the factory class.

```cpp
// Provides Vertica with metadata about the StringTokenizer
class TokenFactory : public TransformFunctionFactory {
    // Tell Vertica that StringTokenizer reads 1 int and 1 string,
    // and returns two ints and a string
    virtual void getPrototype(ServerInterface &srvInterface, ColumnTypes &argTypes, ColumnTypes &returnType) {
        argTypes.addInt(); // Row number
        argTypes.addVarchar(); // Line of text
        returnType.addInt(); // The token
        returnType.addInt(); // The row this token occurred in
        returnType.addInt(); // The position in the row of the token
    }

    // Gives details of the output columns. For integer rows,
    // Tell Vertica the maximum return string length will be, given the input
    // string length. Also names the output columns. This function is only
    // necessary if you have a column that has a variable size (i.e. strings) or
    // have to report their precision.
    virtual void getReturnType(ServerInterface &srvInterface, const SizedColumnTypes &input_types,
        ColumnTypes &argTypes, ColumnTypes &returnType, ColumnTypes &inputPrototypes)
```
In this example:

- The UDTF takes an INTEGER and a VARCHAR column as input. To define these input columns, getPrototype() calls addInt() and addVarchar() on the ColumnTypes object that represents the input table.

- The UDTF returns a VARCHAR and two INTEGER columns as output. The getPrototype() function calls addVarchar() once and addInt() twice to define the output table.

This example must return the maximum length of the VARCHAR output column. It sets the length to the length input string. This is a safe value, because the output will never be longer than the input string. It also sets the name of the VARCHAR output column to "token". You can use getReturnType() to name the output columns, even if they do not require a precision or output length. The example sets the names of the two INTEGER output columns to "row_id" and "token_position".

Note: You are not required to supply a name for an output column in this function. However, it is a best practice to do so. If you do not name an output column, getReturnType() sets the column name to ". The SQL statements that call your UDTF must provide aliases for any unnamed columns to access them or else they return an error. From a usability standpoint, it is easier for you to supply the column names here once. The alternative is to force all of the users of your function to supply their own column names for each call to the UDTF.

The implementation of the createTransformFunction() function in the example is boilerplate code. It just calls the vt_returnFuncObj macro with the name of the TransformFunction class associated with this factory class. This macro takes care of instantiating a copy of the TransformFunction class that Vertica can use to process data.
The RegisterFactory Macro

The final step in creating your UDTF is to call the RegisterFactory macro. This macro ensures that your factory class is instantiated when Vertica loads the shared library containing your UDTF. Having your factory class instantiated is the only way that Vertica can find your UDTF and determine what its inputs and outputs are.

The RegisterFactory macro just takes the name of your factory class:

```cpp
RegisterFactory(TokenFactory);
```

C++ Example: Multi-Phase Transform Function

The following code fragment is from the InvertedIndex UDTF example distributed with the Vertica SDK. It demonstrates subclassing the MultiPhaseTransformFunctionFactory including two TransformFunctionPhase subclasses that define the two phases in this UDTF.

```cpp
class InvertedIndexFactory : public MultiPhaseTransformFunctionFactory
{
public:
  /**
   * Extracts terms from documents.
   */
  class ForwardIndexPhase : public TransformFunctionPhase
  {
    virtual void getReturnType(ServerInterface &srvInterface,
                                const SizedColumnTypes &inputTypes,
                                SizedColumnTypes &outputTypes)
    {
      // Sanity checks on input we've been given.
      // Expected input: (doc_id INTEGER, text VARCHAR)
      vector<size_t> argCols;
      inputTypes.getArgumentColumns(argCols);
      if (argCols.size() < 2 ||
          !inputTypes.getColumnType(argCols.at(0)).isInt() ||
          !inputTypes.getColumnType(argCols.at(1)).isVarchar())
        vt_report_error(0, "Function only accepts two arguments"
                        "(INTEGER, VARCHAR)");
      // Output of this phase is:
      // (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
      // Number of times term appears within a document.
      outputTypes.addInt("term_freq");
      // Add analytic clause columns: (PARTITION BY term ORDER BY doc_id).
      // The length of any term is at most the size of the entire document.
      outputTypes.addVarcharPartitionColumn(
        inputTypes.getColumnType(argCols.at(1)).getStringLength(),
        "term");
      // Add order column on the basis of the document id's data type.
      outputTypes.addOrderColumn(inputTypes.getColumnType(argCols.at(0)),
                                 "doc_id");
    }
```
virtual TransformFunction *createTransformFunction(ServerInterface &srvInterface)
{
    return vt_createFuncObj(srvInterface.allocation, ForwardIndexBuilder); }
};

/**
* Constructs terms' posting lists.
*/
class InvertedIndexPhase : public TransformFunctionPhase
{
    virtual void getReturnType(ServerInterface &srvInterface,
        const SizedColumnTypes &inputTypes,
        SizedColumnTypes &outputTypes)
    {
        // Sanity checks on input we've been given.
        // Expected input:
        // (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
        // Output of this phase is:
        // (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq INTEGER)
        if (argCols.size() != 1 || pByCols.size() != 1 || oByCols.size() != 1)
        vt_report_error(0, "Function expects an argument (INTEGER) with "
                        "an analytic clause OVER(PBY VARCHAR OBY INTEGER)");
        // Output of this phase is:
        // (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq INTEGER)
        outputTypes.addVarchar(inputTypes.getColumnType(pByCols.at(0)).getStringLength(), "term");
        outputTypes.addInt("doc_id");
        // Number of times term appears within the document.
        outputTypes.addInt("term_freq");
        // Number of documents where the term appears in.
        outputTypes.addInt("corp_freq");
    }
    virtual TransformFunction *createTransformFunction(ServerInterface &srvInterface)
    {
        return vt_createFuncObj(srvInterface.allocation, InvertedIndexBuilder); }
};

ForwardIndexPhase fwardIdxPh;
InvertedIndexPhase invIdxPh;
virtual void getPhases(ServerInterface &srvInterface,
    std::vector<TransformFunctionPhase *> &phases)
{
    fwardIdxPh.setPrepass(); // Process documents wherever they're originally stored.
    phases.push_back(&fwardIdxPh);
    phases.push_back(&invIdxPh);
    }
Most of the code in this example is similar to the code in a TransformFunctionFactory class:

- Both TransformFunctionPhase subclasses implement the getReturnType() function, which describes the output of each stage. This is similar to the getReturnType() function from the TransformFunctionFactory class. However, this function also lets you control how the data is partitioned and ordered between each phase of your multi-phase UDTF.

  The first phase calls SizedColumnTypes::addVarcharPartitionColumn() (rather than just addVarcharColumn()) to set the phase's output table to be partitioned by the column containing the extracted words. It also calls SizedColumnTypes::addOrderColumn() to order the output table by the document ID column. It calls this function instead of one of the data-type-specific functions (such as addIntOrderColumn()) so it can pass the data type of the original column through to the output column.

  **Note:** Any order by column or partition by column set by the final phase of the UDTF in its getReturnType() function is ignored. Its output is returned to the initiator node rather than partitioned and reordered then sent to another phase.

- The MultiPhaseTransformFunctionFactory class implements the getPrototype() function, that defines the schemas for the input and output of the multi-phase UDTF. This function is the same as the TransformFunctionFactory::getPrototype() function.

  The unique function implemented by the MultiPhaseTransformFunctionFactory class is getPhases(). This function defines the order in which the phases are executed. The fields that represent the phases are pushed into this vector in the order they should execute.

  The MultiPhaseTransformFunctionFactory::getPhases() function is also where you flag the first phase of the UDTF as operating on data stored locally on the node (called a "pre-pass" phase) rather than on data partitioned across all nodes. Using this option increases the efficiency of your multi-phase UDTF by avoiding having to move significant amounts of data around the Vertica cluster.
Note: Only the first phase of your UDTF can be a pre-pass phase. You cannot have multiple pre-pass phases, and no later phase can be a pre-pass phase.

To mark the first phase as pre-pass, you call the `TransformFunctionPhase::setPrepass()` function of the first phase's `TransformFunctionPhase` instance from within the `getPhase()` function.

Notes

- You need to ensure that the output schema of each phase matches the input schema expected by the next phase. In the example code, each `TransformFunctionPhase::getReturnType()` implementation performs a sanity check on its input and output schemas. Your `TransformFunction` subclasses can also perform these checks in their `processPartition()` function.

- There is no built-in limit on the number of phases that your multi-phase UDTF can have. However, more phases use more resources. When running in fenced mode, Vertica may terminate UDTFs that use too much memory. See Resource Use for C++ UDx.

Java API

This section provides APIs and examples for the Java API for UDTFs.

For information on setting up a Java development environment and compiling and packaging libraries, see Developing with the Java SDK.

TransformFunction and TransformFunctionFactory

Java Interface

This section describes information that is specific to the Java API. See UDTF Class Overview for general information about implementing the `TransformFunction` and `TransformFunctionFactory` classes.

TransformFunction API

The API provides the following methods for extension by subclasses:

```java
void setup(ServerInterface srvInterface, SizedColumnTypes argTypes);
abstract void processPartition(ServerInterface srvInterface,
```
PartitionReader input_reader, PartitionWriter input_writer) throws UdfException, DestroyInvocation;

void destroy(ServerInterface srvInterface, SizedColumnTypes argTypes);

TransformFunctionFactory API

The API provides the following methods for extension by subclasses:

abstract TransformFunction createTransformFunction(ServerInterface srvInterface);

abstract void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType);

abstract void getReturnType(ServerInterface srvInterface, SizedColumnTypes argTypes,
  SizedColumnTypes returnType) throws UdfException;

void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);

MultiPhaseTransformFunctionFactory API

The MultiPhaseTransformFunctionFactory class extends TransformFunctionFactory The API provides the following additional methods for extension by subclasses:

abstract void getPhases(ServerInterface srvInterface, Vector<TransformFunctionPhase> phases);

If using this factory you must also extend TransformFunctionPhase. See the SDK reference documentation.

Java Example: String Tokenizer

The following example shows a subclass of TransformFunction named StringTokenizer. It defines a UDTF that reads a table containing an INTEGER ID column and a VARCHAR column. It breaks the text in the VARCHAR column into tokens (individual words). It returns a table containing each token, the row it occurred in, and its position within the string.

You can then use it from SQL statements, for example:

=> CREATE TABLE T (url varchar(30), description varchar(2000));
CREATE TABLE
=> INSERT INTO T VALUES ('www.amazon.com','Online retail merchant and provider of cloud services');
OUTPUT
----------
1
(1 row)
=> INSERT INTO T VALUES ('www.hp.com','Leading provider of computer hardware and imaging solutions');
OUTPUT
----------
1
(1 row)
=> INSERT INTO T VALUES ('www.vertica.com','World''s fastest analytic database');
OUTPUT
----------
1
(1 row)
=> COMMIT;
COMMIT
=> -- Invoke the UDT
=> SELECT url, tokenize(description) OVER (partition by url) FROM T;

<table>
<thead>
<tr>
<th>url</th>
<th>words</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>Online</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>retail</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>merchant</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>and</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>provider</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>of</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>c</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>loud</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>services</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>Leading</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>provider</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>of</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>computer</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>hardware</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>and</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>im</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>aging</td>
</tr>
<tr>
<td><a href="http://www.hp.com">www.hp.com</a></td>
<td>solutions</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>World's</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>fastest</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>analytic</td>
</tr>
<tr>
<td><a href="http://www.vertica.com">www.vertica.com</a></td>
<td>database</td>
</tr>
</tbody>
</table>

(22 rows)

TransformFunction Implementation

The following code defines the StringTokenizer class.

To make code management simpler, the TransformFunction class is defined as an inner class of the TransformFactoryClass.

// You will need to specify the full package when creating functions based on // the classes in your library.
package com.mycompany.example;
// Import the entire Vertica SDK
import com.vertica.sdk.*;

public class TokenFactory extends TransformFunctionFactory
{
    // Set the number and data types of the columns in the input and output rows.
    @Override
    public void getPrototype(ServerInterface srvInterface, ColumnTypes argTypes, ColumnTypes returnType)
    {
        // Define two input columns: an INTEGER and a VARCHAR
        argTypes.addInt(); // Row id
        argTypes.addVarchar(); // Line of text

        // Define the output columns
        returnType.addInt(); // The token
        returnType.addInt(); // The row in which this token occurred
        returnType.addInt(); // The position in the row of the token
    }

    // Set the width of any variable-width output columns, and also name
    // them.
    @Override
    public void getReturnType(ServerInterface srvInterface, SizedColumnTypes inputTypes, SizedColumnTypes outputTypes)
    {
        // Set the maximum width of the token return column to the width
        // of the input text column and name the output column “Token”
        outputTypes.addVarchar(inputTypes.getColumnType(1).getStringLength(), "token");

        // Name the two INTEGER output columns
        outputTypes.addInt("row_id");
        outputTypes.addInt("token_position");
    }

    // Inner class that actually performs work.
    public class TokenizeString extends TransformFunction
    {
        @Override
        public void processPartition(ServerInterface srvInterface, PartitionReader inputReader, PartitionWriter outputWriter)
        throws UdfException, DestroyInvocation
        {
            try {
                // Loop over all rows passed in in this partition.
                do {
                    // Test if the row ID is null. If so, then do not
                    // process any further. Skip to next row of input.
                    if(inputReader.isNull(0)) {
                        srvInterface.log("Skipping row with null id.");
                        continue; // Move on to next row of input
                    }

                    // Get the row ID now that we know it has a value
                    long rowId = inputReader.getLong(0);

                    // Test if the input string is NULL. If so, return NULL
                    // for token and string position.
                    if (inputReader.isNull(1)) {
                        outputWriter.setStringNull(0);
                    }
                }
            }
        }
    }
}
outputWriter.setLong(1, rowId);
outputWriter.setLongNull(2);
outputWriter.next(); // Move to next line of output
} else {
    // Break string into tokens. Output each word as its own
    // value.
    String[] tokens = inputReader.getString(1).split("\s+");
    // Output each token on a separate row.
    for (int i = 0; i < tokens.length; i++) {
        outputWriter.getStringWriter(0).copy(tokens[i]);
        outputWriter.setLong(1, rowId);
        outputWriter.setLong(2, i);
        outputWriter.next(); // Advance to next row of output
    }
    // Loop until there are no more input rows in partition.
} while (inputReader.next());
} // Prevent exceptions from bubbling back up to server. Uncaught
// exceptions will cause a transaction rollback.
catch (Exception e) {
    // Use more robust error handling in your own
    // UDTFs. This example just sends a message to the log.
    srvInterface.log("Exception: " + e.getClass().getSimpleName()
        + "Message: " + e.getMessage());
}
}

@Override
public TransformFunction createTransformFunction(ServerInterface srvInterface)
{
    return new TokenizeString();
}

---

Java Example: Multi-Phase Transform Function

The following code is excerpted from the InvertedIndexFactory SDK example. You can find the complete code in
/opt/vertica/sdk/examples/JavaUDx/TransformFunctions.

```java
public class InvertedIndexFactory extends MultiPhaseTransformFunctionFactory {
    public class ForwardIndexPhase extends TransformFunctionPhase {
        // ...
    }

    public class InvertedIndexPhase extends TransformFunctionPhase {

        @Override
        public TransformFunction createTransformFunction(ServerInterface srvInterface) {
            return new InvertedIndexBuilder();
        }
    }
}
```
public void getReturnType(ServerInterface srvInterface,
SizedColumnTypes inputTypes,
SizedColumnTypes outputTypes) {
// Sanity checks on input we've been given.
// Expected input:
// (term_freq INTEGER) OVER(PBY term VARCHAR OBY doc_id INTEGER)
ArrayList<Integer> argCols = new ArrayList<Integer>();
inputTypes.getArgumentColumns(argCols);
ArrayList<Integer> pByCols = new ArrayList<Integer>();
inputTypes.getPartitionByColumns(pByCols);
ArrayList<Integer> oByCols = new ArrayList<Integer>();
inputTypes.getOrderByColumns(oByCols);
if (argCols.size() != 1 || pByCols.size() != 1
|| oByCols.size() != 1 ||
!inputTypes.getColumnType(argCols.get(0)).isInt()
|| !inputTypes.getColumnType(pByCols.get(0)).isVarchar()
|| !inputTypes.getColumnType(oByCols.get(0)).isInt()) {
    throw new UdfException(0,
        "Function expects an argument (INTEGER) with "
        + "analytic clause OVER(PBY VARCHAR OBY INTEGER)");
}

// Output of this phase is:
// (term VARCHAR, doc_id INTEGER, term_freq INTEGER, corp_freq INTEGER)
outputTypes.addVarchar(inputTypes.getColumnType(pByCols.get(0)).
    getStringLength(), "term");
outputTypes.addInt("doc_id");

// Number of times term appears within the document.
outputTypes.addInt("term_freq");

// Number of documents where the term appears in.
outputTypes.addInt("corp_freq");
}

@Override
public void getPhases(ServerInterface srvInterface,
Vector<TransformFunctionPhase> phases) {
    ForwardIndexPhase fwardIdxPh;
    InvertedIndexPhase invIdxPh;
    fwardIdxPh = new ForwardIndexPhase();
    invIdxPh = new InvertedIndexPhase();
    fwardIdxPh.setPrepass();
    phases.add(fwardIdxPh);;
    phases.add(invIdxPh);
}

@Override
public void getPrototype(ServerInterface srvInterface,
ColumnTypes argTypes, ColumnTypes returnTypes) {

Most of the code in this example is similar to the code in a TransformFunctionFactory class. The getReturnType method is similar, but in a TransformFunctionPhase it can also partition and order the data. The partitions and order are used by the next phase, but they are ignored after the final phase.

The MultiPhaseTransformFunctionFactory class adds the getPhases method. This method defines the order in which the phases are executed. The fields that represent the phases are pushed into this vector in the order they should execute. You should also indicate the first phase by calling setPrepass() on it. Doing so can lead to better performance.

There is no built-in limit on the number of phases that your multi-phase UDTF can have. However, more phases use more resources. Vertica might terminate UDTFs that use too much memory.

R API

This section provides APIs and examples for the R API for UDTFs.

For information on setting up an R development environment and compiling and packaging libraries, see Developing with the R SDK.

Transform Function and Transform Function Factory R Interface

This section describes information that is specific to the R API. See R SDK API Documentation for general information about implementing transform functions.

Transform Function API

The API provides the following framework for extension by R function:
FunctionName <- function(input.data.frame, parameters.data.frame) {
    # Computations
    # The function must return a data frame.
    return(output.data.frame)
}

Transform Function Factory API

The API provides the following framework for extension by R function:

FunctionNameFactory <- function() {
    list(name = FunctionName,
         udxtype = c("transform"),
         intype = c("int"),
         outtype = c("int"))
}

R Example: Log Tokenizer

The LogTokenizer transform function reads a varchar from a table, a log message. It then
tokenizes each of the log messages, returning each of the tokens.

You can find more UDx examples in the VerticaGithub repository,

Load the Function and Library

Create the library and the function.

=> CREATE OR REPLACE LIBRARY rLib AS 'log_tokenizer.R' LANGUAGE 'R';
CREATE LIBRARY
=> CREATE OR REPLACE TRANSFORM FUNCTION LogTokenizer AS LANGUAGE 'R' NAME 'LogTokenizerFactory'
LIBRARY rLib FENCED;
CREATE FUNCTION

Querying Data with the Function

The following query shows how you can run a query with the UDTF.

=> SELECT machine,
    LogTokenizer(error_log USING PARAMETERS spliton = ' ') OVER(PARTITION BY machine)
FROM error_Logs;

machine | Token
---------
node001 | ERROR
```
UDTF R Code

LogTokenizer <- function(input.data.frame, parameters.data.frame) {
  # Take the spliton parameter passed by the user and assign it to a variable
  # in the function so we can use that as our tokenizer.
  if (is.null(parameters.data.frame[['spliton']])) {
    stop("NULL value for spliton! Token cannot be NULL.")
  } else {
    split.on <- as.character(parameters.data.frame[['spliton']])
  }
  # Tokenize the string.
  tokens <- vector(length=0)
  for ( string in input.data.frame[, 1] ) {
    tokenized.string <- strsplit(string, split.on)
    for ( token in tokenized.string ) {
      tokens <- append(tokens, token)
    }
  }
  final.output <- data.frame(tokens)
  return(final.output)
}

LogTokenizerFactory <- function() {
  list(name = LogTokenizer,
       udxtype = c("transform"),
       intype = c("varchar"),
       outtype = c("varchar"),
       outtypecallback=LogTokenizerReturn,
       parametertypecallback=LogTokenizerParameters)
}

LogTokenizerParameters <- function() {
  parameters <- list(datatype = c("varchar"),
```
LogTokenizerReturn <- function(arg.data.frame, parm.data.frame) {
  output.return.type <- data.frame(
    datatype = rep(NA,1),
    length = rep(NA,1),
    scale = rep(NA,1),
    name = rep(NA,1))
  output.return.type$datatype <- c("varchar")
  output.return.type$name <- c("Token")
  return(output.return.type)
}
Load (UDLs)

The COPY statement offers extensive options and settings to control how to load data. However, you may find that these options do not suit the type of data load that you want to perform. The User-Defined Load (UDL) feature lets you develop one or more functions that change how the COPY statement operates. You can create custom libraries using the Vertica SDK to handle various steps in the loading process. The SDK provides APIs for C++ and Java.

You use three types of UDL functions during development, one for each stage of the data-load process:

- **User-Defined Source** (UDSource): Controls how the COPY statement obtains the data it loads into the database. For example, COPY might obtain data by fetching it through HTTP or cURL. Up to one UDSource reads data from a file or input stream. Your UDSource can read from more than one source, but COPY invokes only one UDSource.

- **User-Defined Filter** (UDFilter): Preprocesses the data. For example, a filter might unzip a file or convert UTF-16 to UTF-8. You can chain multiple User-Defined Filters together, for example unzipping and then converting.

- **User-Defined Parser** (UDParser): Up to one parser parses the data into tuples that are ready to be inserted into a table. For example, a parser could extract data from an XML-like format. You can optionally define a User-Defined Chunker (UDChunker, C++ only), to have the parser perform parallel parsing.

After the final step, COPY inserts the data into a table, or rejects it, if it is not in the correct format.

Important: You cannot use fenced-mode UDLs developed in different programming languages together in the same COPY statement. For example, you cannot have a User-Defined Source written in C++ send data to a User-Defined Filter developed in Java. Attempting to do so results in a "Failure in UDx RPC call" error message.

User-Defined Source

A User-Defined Source allows you to process a source of data using a method that is not built into Vertica. For example, you can write a User-Defined Source to access the data from an HTTP source using cURL. While you can use only a single User-Defined Source in a COPY statement, that source function can pull data from multiple sources.
The UDSource class is responsible for acquiring data from an external source. It reads data from an input stream and produces an output stream to be filtered and parsed. If you implement a UDSource, you must also implement a corresponding SourceFactory. See UDSource Class and SourceFactory Class for API details.

UDSource Class

You can subclass the UDSource class when you need to load data from a source type that COPY does not already support.

Each instance of your UDSource subclass reads from a single data source. Examples of a single data source are a single file or the results of a single function call to a RESTful web application.

UDSource Methods

The UDSource class defines the following methods. Your subclass must override process(). You can optionally override the others.

For the signatures of these methods and language-specific information, see Source Classes (C++) and Source Classes (Java).

Setting Up

COPY calls setup() before the first time it calls process(). Use setup() to perform any necessary setup steps to access the data source. This method establishes network connections, opens files, and similar tasks that need to be performed before the UDSource can read data from the data source. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

Processing a Source

COPY calls process() repeatedly during query execution to read data and write it to the DataBuffer passed as a parameter. This buffer is then passed to the first filter.

If the source runs out of input, or fills the output buffer, it must return the value StreamState.OUTPUT_NEEDED. When Vertica gets this return value, it will call the method again. This second call occurs after the output buffer has been processed by the next stage in
the data-load process. Returning StreamState.DONE indicates that all of the data from the source has been read.

The user can cancel the load operation, which aborts reading.

## Tearing Down

COPY calls destroy() after the last time that process() is called. This method frees any resources reserved by the setup() or process() methods, such as file handles or network connections that the setup() method allocated.

## Accessors

A source can define two accessors, getSize() and getUri().

COPY might call getSize() to estimate the number of bytes of data to be read before calling process(). This value is an estimate only and is used to indicate the file size in the LOAD_STREAMS table. Because Vertica can call this method before calling setup(), getSize() must not rely on any resources allocated by setup().

This method should not leave any resources open. For example, do not save any file handles opened by getSize() for use by the process() method. Doing so can exhaust the available resources, because Vertica calls getSize() on all instances of your UDSource subclass before any data is loaded. If many data sources are being opened, these open file handles could use up the system’s supply of file handles. Thus, none would remain available to perform the actual data load.

Vertica calls getUri() during execution to update status information about which resources are currently being loaded. It returns the URI of the data source being read by this UDSource.

## SourceFactory Class

If you write a source, you must also write a source factory. Your subclass of the SourceFactory class is responsible for:

- Performing the initial validation of the parameters passed to your UDSource.

- Setting up any data structures your UDSource instances need to perform their work. This information can include recording which nodes will read which data source.
Creating one instance of your UDSource subclass for each data source (or portion thereof) that your function reads on each host.

The simplest source factory creates one UDSource instance per data source per executor node. You can also use multiple concurrent UDSource instances on each node. This behavior is called concurrent load. To support both options, SourceFactory has two versions of the method that creates the sources. You must implement exactly one of them.

Source factories are singletons. Your subclass must be stateless, with no fields containing data. The subclass also must not modify any global variables.

SourceFactory Methods

The SourceFactory class defines several methods. Your class must override prepareUDSources(); it may override the other methods.

Setting Up

Vertica calls plan() once on the initiator node to perform the following tasks:

- Check the parameters the user supplied to the function call in the COPY statement and provide error messages if there are any issues. You can read the parameters by getting a ParamReader object from the instance of ServerInterface passed into the plan() method.

- Decide which hosts in the cluster will read the data source. How you divide up the work depends on the source your function is reading. Some sources can be split across many hosts, such as a source that reads data from many URLs. Others, such as an individual local file on a host's filesystem, can be read only by a single specified host.

  You store the list of hosts to read the data source by calling the setTargetNodes() method on the NodeSpecifyingPlanContext object. This object is passed into your plan() method.

- Store any information that the individual hosts need to process the data sources in the NodeSpecifyingPlanContext instance passed to the plan() method. For example, you could store assignments that tell each host which data sources to process. The plan() method runs only on the initiator node, and the prepareUDSources() method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.
You store data in the NodeSpecifyingPlanContext by getting a ParamWriter object from the getWriter() method. You then write parameters by calling methods on the ParamWriter such as setString().

Note: ParamWriter offers the ability to store only simple data types. For complex types, you must serialize the data in some manner and store it as a string or long string.

Creating Sources

Vertica calls prepareUDSources() on all hosts that the plan() method selected to load data. This call instantiates and returns a list of UDSource subclass instances. If you are not using concurrent load, return one UDSource for each of the sources that the host is assigned to process. If you are using concurrent load, use the version of the method that takes an ExecutorPlanContext as a parameter, and return as many sources as you can use. Your factory must implement exactly one of these methods.

Note: In the C++ API, the function that supports concurrent load is named prepareUDSourcesExecutor(). In the Java API the class provides two overloads of prepareUDSources().

For concurrent load, you can find out how many threads are available on the node to run UDSource instances by calling getLoadConcurrency() on the ExecutorPlanContext that is passed in.

Defining Parameters

Implement getParameterTypes() to define the names and types of parameters that your source uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.
Requesting Threads for Concurrent Load

When a source factory creates sources on an executor node, by default, it creates one thread per source. If your sources can use multiple threads, implement `getDesiredThreads()`. Vertica calls this method before it calls `prepareUDSources()`, so you can also use it to decide how many sources to create. Return the number of threads your factory can use for sources. The maximum number of available threads is passed in, so you can take that into account. The value your method returns is a hint, not a guarantee; each executor node determines the number of threads to allocate. The `FilePortionSourceFactory` example implements this method; see Source Example: Concurrent Load.

You can allow your source to have control over parallelism, meaning that it can divide a single input into multiple load streams, by implementing `isSourceApportionable()`. Returning `true` from this method does not guarantee that the source will apportion the load. However, returning `false` indicates that it will not try to do so. See Apportioned Load for more information.

Often, a `SourceFactory` that implements `getDesiredThreads()` also uses apportioned load. However, using apportioned load is not a requirement. A source reading from Kafka streams, for example, could use multiple threads without apportioning.

User-Defined Filter

User-Defined Filter functions allow you to manipulate data obtained from a source in various ways.

For example, a filter could:

- Process a compressed file in a compression format not natively supported by Vertica.
- Take UTF-16-encoded data and transcode it to UTF-8 encoding.
- Perform search-and-replace operations on data before it is loaded into Vertica.

You can also process data through multiple filters before it is loaded into Vertica. For instance, you could unzip a file compressed with GZip, convert the content from UTF-16 to UTF-8, and finally search and replace certain text strings.

If you implement a `UDFilter`, you must also implement a corresponding `FilterFactory`. See `UDFilter Class` and `FilterFactory Class` for API details.
UDFilter Class

The UDFilter class is responsible for reading raw input data from a source and preparing it to be loaded into Vertica or processed by a parser. This preparation may involve decompression, re-encoding, or any other sort of binary manipulation.

A UDFilter is instantiated by a corresponding FilterFactory on each host in the Vertica cluster that is performing filtering for the data source.

UDFilter Methods

The UDFilter class defines the following methods. Your subclass must override process(). You can optionally override the other methods.

For the signatures of these methods and language-specific information, see Filter Classes (C++) and Filter Classes (Java).

Setting Up

COPY calls setup() before the first time it calls process(). Use setup() to perform any necessary setup steps that your filter needs to operate, such as initializing data structures to be used during filtering. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

Filtering Data

COPY calls process() repeatedly during query execution to filter data. The method receives two instances of the DataBuffer class among its parameters, an input and an output buffer. Your implementation should read from the input buffer, manipulate it in some manner (such as decompressing it), and write the result to the output. A one-to-one correlation between the number of bytes your implementation reads and the number it writes might not exist. The process() method should process data until it either runs out of data to read or runs out of space in the output buffer. When one of these conditions occurs, your method should return one of the following values defined by StreamState:
- **OUTPUT_NEEDED** if the filter needs more room in its output buffer.
- **INPUT_NEEDED** if the filter has run out of input data (but the data source has not yet been fully processed).
- **DONE** if the filter has processed all of the data in the data source.
- **KEEP_GOING** if the filter cannot proceed for an extended period of time. The method will be called again, so do not block indefinitely. If you do, then you prevent your user from canceling the query.

Before returning, your `process()` method must set the `offset` property in each `DataBuffer`. In the input buffer, set it to the number of bytes that the method successfully read. In the output buffer, set it to the number of bytes the method wrote. Setting these properties allows the next call to `process()` to resume reading and writing data at the correct points in the buffers.

Your `process()` method also needs to check the `InputState` object passed to it to determine if there is more data in the data source. When this object is equal to `END_OF_FILE`, then the data remaining in the input data is the last data in the data source. Once it has processed all of the remaining data, `process()` must return **DONE**.

## Tearing Down

COPY calls `destroy()` after the last time it calls `process()`. This method frees any resources reserved by the `setup()` or `process()` methods. Vertica calls this method after the `process()` method indicates it has finished filtering all of the data in the data stream.

If there are still data sources that have not yet been processed, Vertica may later call `setup()` on the object again. On subsequent calls Vertica directs the method to filter the data in a new data stream. Therefore, your `destroy()` method should leave an object of your `UDFilter` subclass in a state where the `setup()` method can prepare it to be reused.

### FilterFactory Class

If you write a filter, you must also write a filter factory to produce filter instances. To do so, subclass the `FilterFactory` class.

Your subclass performs the initial validation and planning of the function execution and instantiates `UDFilter` objects on each host that will be filtering data.
Filter factories are singletons. Your subclass must be stateless, with no fields containing data. The subclass also must not modify any global variables.

FilterFactory Methods

The FilterFactory class defines the following methods. Your subclass must override the prepare() method. It may override the other methods.

Setting Up

Vertica calls plan() once on the initiator node, to perform the following tasks:

- Check any parameters that have been passed from the function call in the COPY statement and error messages if there are any issues. You read the parameters by getting a ParamReader object from the instance of ServerInterface passed into your plan() method.

- Store any information that the individual hosts need in order to filter data in the PlanContext instance passed as a parameter. For example, you could store details of the input format that the filter will read and output the format that the filter should produce. The plan() method runs only on the initiator node, and the prepare() method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.

You store data in the PlanContext by getting a ParamWriter object from the getWriter() method. You then write parameters by calling methods on the ParamWriter such as setString.

Note: ParamWriter offers only the ability to store simple data types. For complex types, you need to serialize the data in some manner and store it as a string or long string.

Creating Filters

Vertica calls prepare() to create and initialize your filter. It calls this method once on each node that will perform filtering. Vertica automatically selects the best nodes to complete the work based on available resources. You cannot specify the nodes on which the work is done.
Defining Parameters

Implement `getParameterTypes()` to define the names and types of parameters that your filter uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.

User-Defined Parser

A parser takes a stream of bytes and passes a corresponding sequence of tuples to the Vertica load process. You can use User-Defined Parser functions to parse:

- Data in formats not understood by the Vertica built-in parser.
- Data that requires more specific control than the built-in parser supplies.

For example, you could load a CSV file using a specific CSV library. See the Vertica SDK for two CSV examples.

COPY supports a single User-Defined Parser that you can use with a UDataSource and zero or more instances of UDFilter.

Sometimes you can improve the performance of your parser by adding a chunker. A chunker divides up the input and uses multiple threads to parse it. See Cooperative Parse. Chunkers are available only in the C++ API.

Under special circumstances you can further improve performance by using apportioned load, an approach where multiple Vertica nodes parse the input. See Apportioned Load.

If you implement a UDParser, you must also implement a corresponding ParserFactory. See UDParser Class and ParserFactory Class for API details.

UDParser Class

You can subclass the UDParser class when you need to parse data that is in a format that the COPY statement's native parser cannot handle.
During parser execution, Vertica always calls three methods: `setup()`, `process()`, and `destroy()`. It might also call `getRejectedRecord()`.

**UDParser Methods**

The `UDParser` class defines the following methods. Your subclass must override the `process()` and `getRejectedRecord()` methods. You can optionally override the other methods.

For the signatures of these methods and language-specific information, see Parser Classes (C++) and Parser Classes (Java).

**Note:** The `UDParser` class performs important initialization required by all subclasses, including initializing the `StreamWriter` object used by the parser. Therefore, your constructor must call `super()`.

**Setting Up**

COPY calls `setup()` before the first time it calls `process()`. Use `setup()` to perform any initial setup tasks that your parser needs to parse data. This setup includes retrieving parameters from the class context structure or initializing data structures for use during filtering. Vertica calls this method before calling the `process()` method for the first time. Your object might be destroyed and re-created during use, so make sure that your object is restartable.

**Parsing**

COPY calls `process()` repeatedly during query execution. Vertica passes this method a buffer of data to parse into columns and rows and one of the following input states defined by `InputState`:

- **OK**: currently at the start of or in the middle of a stream
- **END_OF_FILE**: no further data is available.
- **END_OF Chunk**: the current data ends on a record boundary and the parser should consume all of it before returning. This input state only occurs when using a chunker.
- **START_OF_PORTION**: the input does not start at the beginning of a source. The parser should find the first end-of-record mark. This input state only occurs when using apportioned load. You can use the `getPortion()` method to access the offset and size of the portion.

- **END_OF_PORTION**: the source has reached the end of its portion. The parser should finish processing the last record it started and advance no further. This input state only occurs when using apportioned load.

The parser must reject any data that it cannot parse, so that Vertica can report the rejection and write the rejected data to files.

The `process()` method must parse as much data as it can from the input buffer. The buffer might not end on a row boundary. Therefore, it might have to stop parsing in the middle of a row of input and ask for more data. The input can contain null bytes, if the source file contains them, and is not automatically null-terminated.

A parser has an associated `StreamWriter` object, which performs the actual writing of the data. When your parser extracts a column value, it uses one of the type-specific methods on `StreamWriter` to write that value to the output stream. See [Writing Data](#) for more information about these methods.

A single call to `process()` might write several rows of data. When your parser finishes processing a row of data, it must call `next()` on its `StreamWriter` to advance the output stream to a new row. (Usually a parser finishes processing a row because it encounters an end-of-row marker.)

When your `process()` method reaches the end of the buffer, it tells Vertica its current state by returning one of the following values defined by `StreamState`:

- **INPUT_NEEDED**: the parser has reached the end of the buffer and needs more data to parse.

- **DONE**: the parser has reached the end of the input data stream.

- **REJECT**: the parser has rejected the last row of data it read (see [Rejecting Rows](#)).

## Tearing Down

COPY calls `destroy()` after the last time that `process()` is called. It frees any resources reserved by the `setup()` or `process()` method.
Vertica calls this method after the `process()` method indicates it has completed parsing the data source. However, sometimes data sources that have not yet been processed might remain. In such cases, Vertica might later call `setup()` on the object again and have it parse the data in a new data stream. Therefore, write your `destroy()` method so that it leaves an instance of your `UDParser` subclass in a state where `setup()` can be safely called again.

## Reporting Rejections

If `process()` rejects a row, Vertica calls `getRejectedRecord()` to report it. Usually, this method returns an instance of the `RejectedRecord` class with details of the rejected row.

### Writing Data

A parser has an associated `StreamWriter` object, which you access by calling `getStreamWriter()`. In your `process()` implementation, use the `setType()` methods on the `StreamWriter` object to write values in a row to specific column indexes. Verify that the data types you write match the data types expected by the schema.

The following example shows how you can write a value of type `long` to the fourth column (index 3) in the current row:

```java
StreamWriter writer = getStreamWriter();
...
writer.setLongValue(3, 98.6);
```

`StreamWriter` provides methods for all the basic types, such as `setBooleanValue()`, `setStringValue()`, and so on. See the API documentation for a complete list of `StreamWriter` methods, including options that take primitive types or explicitly set entries to null.

The Java API supports additional options for writing data. See Parser Classes.

### Rejecting Rows

If your parser finds data it cannot parse, it should reject the row by:

1. Saving details about the rejected row data and the reason for the rejection. These pieces of information can be directly stored in a `RejectedRecord` object, or in fields on your `UDParser` subclass, until they are needed.
2. Updating the row's position in the input buffer by updating `input.offset` so it can resume parsing with the next row.

3. Signaling that it has rejected a row by returning with the value `StreamState.REJECT`.

4. Returning an instance of the `RejectedRecord` class with the details about the rejected row.

Breaking Up Large Loads

Vertica provides two ways to break up large loads. Apportioned Load allows you to distribute a load among several database nodes. Cooperative Parse (C++ only) allows you to distribute a load among several threads on one node.

UDChunker Class

You can subclass the `UDChunker` class to allow your parser to support Cooperative Parse. This class is available only in the C++ API.

Fundamentally, a `UDChunker` is a very simplistic parser. Like `UDParser`, it has the following three methods: `setup()`, `process()`, and `destroy()`. You must override `process()`; you may override the others. This class has one additional method, `alignPortion()`, which you must implement if you want to enable Apportioned Load for your `UDChunker`.

For the signatures of these methods, see Parser Classes.

Setting Up and Tearing Down

As with `UDParser`, you can define initialization and cleanup code for your chunker. Vertica calls `setup()` before the first call to `process()` and `destroy()` after the last call to `process()`. Your object might be reused amongst multiple load sources, so make sure that `setup()` completely initializes all fields.

Chunking

Vertica calls `process()` to divide an input into chunks that can be parsed independently. The method takes an input buffer and an indicator of the input state:

- **OK**: the input buffer begins at the start of or in the middle of a stream.
- **END_OF_FILE**: no further data is available.
- **END_OF_PORTION**: the source has reached the end of its portion. This state occurs only when using apportioned load.

If the input state is **END_OF_FILE**, the chunker should set the `input.offset` marker to `input.size` and return **DONE**. Returning **INPUT_NEEDED** is an error.

If the input state is **OK**, the chunker should read data from the input buffer and find record boundaries. If it finds the end of at least one record, it should align the `input.offset` marker with the byte after the end of the last record in the buffer and return **CHUNKAligned**. For example, if the input is "abc~def" and "~" is a record terminator, this method should set `input.offset` to 4, the position of "d". If `process()` reaches the end of the input without finding a record boundary, it should return **INPUT_NEEDED**.

You can divide the input into smaller chunks, but consuming all available records in the input can have better performance. For example, a chunker could scan backwards from the end of the input to find a record terminator, which might be the last of many records in the input, and return it all as one chunk without scanning through the rest of the input.

If the input state is **END_OF_PORTION**, the chunker should behave as it does for an input state of **OK**, except that it should also set a flag. When called again, it should find the first record in the next portion and align the chunk to that record.

The input data can contain null bytes, if the source file contains them. The input argument is not automatically null-terminated.

The `process()` method must not block indefinitely. If this method cannot proceed for an extended period of time, it should return **KEEP_GOING**. Failing to return **KEEP_GOING** has several consequences, such as preventing your user from being able to cancel the query.

See **Chunker Example: Delimited Parser and Chunker** for an example of the `process()` method using chunking.

**Aligning Portions**

If your chunker supports apportioned load, implement the `alignPortion()` method. Vertica calls this method one or more times, before calling `process()`, to align the input offset with the beginning of the first complete chunk in the portion. The method takes an input buffer and an indicator of the input state:

- **START_OF_PORTION**: the beginning of the buffer corresponds to the start of the portion. You can use the `getPortion()` method to access the offset and size of the portion.
- **OK**: the input buffer is in the middle of a portion.
- **END_OF_PORTION**: the end of the buffer corresponds to the end of the portion or beyond the end of a portion.

- **END_OF_FILE**: no further data is available.

The method should scan from the beginning of the buffer to the start of the first complete record. It should set `input.offset` to this position and return one of the following values:

- **DONE**, if it found a chunk. `input.offset` is the first byte of the chunk.

- **INPUT_NEEDED**, if the input buffer does not contain the start of any chunk. It is an error to return this from an input state of **END_OF_FILE**.

- **REJECT**, if the portion (not buffer) does not contain the start of any chunk.

### ParserFactory Class

If you write a parser, you must also write a factory to produce parser instances. To do so, subclass the **ParserFactory** class.

Parser factories are singletons. Your subclass must be stateless, with no fields containing data. Your subclass also must not modify any global variables.

The **ParserFactory** class defines the following methods. Your subclass must override the **prepare()** method. It may override the other methods.

### Setting Up

Vertica calls **plan()** once on the initiator node to perform the following tasks:

- Check any parameters that have been passed from the function call in the COPY statement and error messages if there are any issues. You read the parameters by getting a **ParamReader** object from the instance of **ServerInterface** passed into your **plan()** method.

- Store any information that the individual hosts need in order to parse the data. For example, you could store parameters in the **PlanContext** instance passed in through the **planCtxt** parameter. The **plan()** method runs only on the initiator node, and the **prepareUDSources()** method runs on each host reading from a data source. Therefore, this object is the only means of communication between them.
You store data in the PlanContext by getting a ParamWriter object from the getWriter() method. You then write parameters by calling methods on the ParamWriter such as setString.

Note: ParamWriter offers only the ability to store simple data types. For complex types, you need to serialize the data in some manner and store it as a string or long string.

Creating Parsers

Vertica calls prepare() on each node to create and initialize your parser, using data stored by the plan() method.

Defining Parameters

Implement getParameterTypes() to define the names and types of parameters that your parser uses. Vertica uses this information to warn callers about unknown or missing parameters. Vertica ignores unknown parameters and uses default values for missing parameters. While you should define the types and parameters for your function, you are not required to override this method.

Defining Parser Outputs

Implement getParserReturnType() to define the data types of the table columns that the parser outputs. If applicable, getParserReturnType() also defines the size, precision, or scale of the data types. Usually, this method reads data types of the output table from the argType and perColumnParamReader arguments and verifies that it can output the appropriate data types. If getParserReturnType() is prepared to output the data types, it calls methods on the SizedColumnTypes object passed in the returnType argument. In addition to the data type of the output column, your method should also specify any additional information about the column's data type:

- For binary and string data types (such as CHAR, VARCHAR, and LONG VARBINARY), specify its maximum length.
- For NUMERIC types, specify its precision and scale.
- For Time/Timestamp types (with or without time zone), specify its precision (-1 means
unspecified).

- For all other types, no length or precision specification is required.

Supporting Cooperative Parse

To support Cooperative Parse, implement prepareChunker() and return an instance of your UDCunker subclass. If isChunkerApportionable() returns true, then it is an error for this method to return null.

Cooperative parse is currently supported only in the C++ API.

Supporting Apportioned Load

To support Apportioned Load, your parser, chunker, or both must support apportioning. To indicate that the parser can apportion a load, implement isParserApportionable() and return true. To indicate that the chunker can apportion a load, implement isChunkerApportionable() and return true.

The isChunkerApportionable() method takes a ServerInterface as an argument, so you have access to the parameters supplied in the COPY statement. You might need this information if the user can specify a record delimiter, for example. Return true from this method if and only if the factory can create a chunker for this input.

Load Parallelism

Vertica can divide the work of loading data, taking advantage of parallelism to speed up the operation. Vertica supports several types of parallelism:

- Distributed load: Vertica distributes files in a multi-file load to several nodes to load in parallel, instead of loading all of them on a single node. Vertica manages distributed load; you do not need to do anything special in your UDL.

- Cooperative parse: A source being loaded on a single node uses multi-threading to parallelize the parse. Cooperative parse divides a load at execution time, based on how threads are scheduled. You must enable cooperative parse in your parser. See Cooperative Parse.

- Apportioned load: Vertica; divides a single large file or other single source into segments, which it assigns to several nodes to load in parallel. Apportioned load divides the load at
planning time, based on available nodes and cores on each node. You must enable apportioned load in your source and parser. See Apportioned Load.

You can support both cooperative parse and apportioned load in the same UDL. Vertica decides which to use for each load operation and might use both. See Combining Cooperative Parse and Apportioned Load.

Cooperative Parse

By default, Vertica parses a data source in a single thread on one database node. You can optionally use cooperative parse to parse a source using multiple threads on a node. More specifically, data from a source passes through a chunker that groups blocks from the source stream into logical units. These chunks can be parsed in parallel. The chunker divides the input into pieces that can be individually parsed, and the parser then parses them concurrently. Cooperative parse is available only for unfenced UDx. (See Fenced Mode.)

To use cooperative parse, a chunker must be able to locate end-of-record markers in the input. Locating these markers might not be possible in all input formats.

Chunks are created by parser factories. At load time, Vertica first calls the UDChunker to divide the input into chunks and then calls the UDParser to parse each chunk.

You can use cooperative parse and apportioned load independently or together. See Combining Cooperative Parse and Apportioned Load.

How Vertica Divides a Load

When Vertica receives data from a source, it calls the chunker's process() method repeatedly. A chunker is, essentially, a lightweight parser; instead of parsing, the process() method divides the input into chunks.

After the chunker has finished dividing the input into chunks, Vertica sends those chunks to as many parsers as are available, calling the process() method on the parser.

Implementing Cooperative Parse

To implement cooperative parse, perform the following actions:

- Subclass UDChunker and implement process().
- In your ParserFactory, implement prepareChunker() to return a UDChunker.
See Chunker Example: Delimited Parser and Chunker for a UDChunker that also supports apportioned load.

**Apportioned Load**

A parser can use more than one database node to load a single input source in parallel. This approach is referred to as *apportioned load*. Some of the parsers built into Vertica support apportioned load.

Apportioned load, like cooperative parse, requires an input that can be divided at record boundaries. The difference is that cooperative parse does a sequential scan to find record boundaries, while apportioned load first jumps (seeks) to a given position and then scans. Some formats, like generic XML, do not support seeking.

To use apportioned load, you must ensure that the source is reachable by all participating database nodes. You typically use apportioned load with distributed file systems.

It is possible for a parser to not support apportioned load directly but to have a chunker that supports apportioning.

You can use apportioned load and cooperative parse independently or together. See Combining Cooperative Parse and Apportioned Load.

**How Vertica Apportions a Load**

If both the parser and its source support apportioning, then you can specify that a single input is to be distributed to multiple database nodes for loading. The SourceFactory breaks the input into portions and assigns them to execution nodes. Each Portion consists of an offset into the input and a size. Vertica distributes the portions and their parameters to the execution nodes. A source factory running on each node produces a UDSource for the given portion.

The UDParser first determines where to start parsing:

- If the portion is the first one in the input, the parser advances to the offset and begins parsing.
- If the portion is not the first, the parser advances to the offset and then scans until it finds the end of a record. Because records can break across portions, parsing begins after the first record-end encountered.

The parser must complete a record, which might require it to read past the end of the portion. The parser is responsible for parsing all records that *begin* in the assigned portion, regardless of where they end. Most of this work occurs within the *process()* method of the parser.
Sometimes, a portion contains nothing to be parsed by its assigned node. For example, suppose you have a record that begins in portion 1, runs through all of portion 2, and ends in portion 3. The parser assigned to portion 1 parses the record, and the parser assigned to portion 3 starts after that record. The parser assigned to portion 2, however, has no record starting within its portion.

If the load also uses Cooperative Parse, then after apportioning the load and before parsing, Vertica divides portions into chunks for parallel loading.

Implementing Apportioned Load

To implement apportioned load, perform the following actions in the source, the parser, and their factories.

In your SourceFactory subclass:

- Implement isSourceApportionable() and return true.

- Implement plan() to determine portion size, designate portions, and assign portions to execution nodes. To assign portions to particular executors, pass the information using the parameter writer on the plan context (PlanContext::getWriter()).

- Implement prepareUDSources(). Vertica calls this method on each execution node with the plan context created by the factory. This method returns the UDSource instances to be used for this node's assigned portions.

- If sources can take advantage of parallelism, you can implement getDesiredThreads() to request a number of threads for each source. See SourceFactory Class for more information about this method.

In your UDSource subclass, implement process() as you would for any other source, using the assigned portion. You can retrieve this portion with getPortion().

In your ParserFactory subclass:

- Implement isParserApportionable() and return true.

- If your parser uses a UDChunker that supports apportioned load, implement isChunkerApportionable().

In your UDParse subclass:

- Write your UDParse subclass to operate on portions rather than whole sources. You can do so by handling the stream states PORTION_START and PORTION_END, or by using the
ContinuousUDParser API. Your parser must scan for the beginning of the portion, find the first record boundary after that position, and parse to the end of the last record beginning in that portion. Be aware that this behavior might require that the parser read beyond the end of the portion.

- Handle the special case of a portion containing no record start by returning without writing any output.

In your UDChunker subclass, implement alignPortion(). See Aligning Portions.

Example

The SDK provides a C++ example of apportioned load in the ApportionLoadFunctions directory:

- FilePortionSource is a subclass of UDSource.
- DelimFilePortionParser is a subclass of ContinuousUDParser.

Use these classes together. You could also use FilePortionSource with the built-in delimited parser.

The following example shows how you can load the libraries and create the functions in the database:

```sql
=> CREATE LIBRARY FilePortionSourceLib as '/home/dbadmin/FP.so';
=> CREATE LIBRARY DelimFilePortionParserLib as '/home/dbadmin/Delim.so';
=> CREATE SOURCE FilePortionSource AS
   LANGUAGE 'C++' NAME 'FilePortionSourceFactory' LIBRARY FilePortionSourceLib;
=> CREATE PARSER DelimFilePortionParser AS
   LANGUAGE 'C++' NAME 'DelimFilePortionParserFactory' LIBRARY DelimFilePortionParserLib;
```

The following example shows how you can use the source and parser to load data:

```sql
=> COPY t WITH SOURCE FilePortionSource(file='g1/*.dat') PARSER DelimFilePortionParser(delimiter = '|', record_terminator = '~');
```

Combining Cooperative Parse and Apportioned Load

You can enable both Cooperative Parse and Apportioned Load in the same parser, allowing Vertica to decide how to load data.
Deciding How to Divide a Load

Vertica uses apportioned load, where possible, at query-planning time. It decides whether to also use cooperative parse at execution time.

Apportioned load requires SourceFactory support. Given a suitable UDSource, at planning time Vertica calls the isParserApportionable() method on the ParserFactory. If this method returns true, Vertica apportions the load.

If isParserApportionable() returns false but isChunkerApportionable() returns true, then a chunker is available for cooperative parse and that chunker supports apportioned load. Vertica apports the load.

If neither of these methods returns true, then Vertica does not apportion the load.

At execution time, Vertica first checks whether the load is running in unfenced mode and proceeds only if it is. Cooperative parse is not supported in fenced mode.

If the load is not apportioned, and more than one thread is available, Vertica uses cooperative parse.

If the load is apportioned, and exactly one thread is available, Vertica uses cooperative parse if and only if the parser is not apportionable. In this case, the chunker is apportionable but the parser is not.

If the load is apportioned, and more than one thread is available, and the chunker is apportionable, Vertica uses cooperative parse.

If Vertica uses cooperative parse but prepareChunker() does not return a UDChunker instance, Vertica reports an error.

Executing Apportioned, Cooperative Loads

If a load uses both apportioned load and cooperative parse, Vertica uses the SourceFactory to break the input into portions. It then assigns the portions to execution nodes. See How Vertica Apportions a Load.

On the execution node, Vertica calls the chunker's alignPortion() method to align the input with portion boundaries. (This step is skipped for the first portion, which by definition is already aligned at the beginning.) This step is necessary because a parser using apportioned load sometimes has to read beyond the end of the portion, so a chunker needs to find the end point.

After aligning the portion, Vertica calls the chunker's process() method repeatedly. See How Vertica Divides a Load.
The chunks found by the chunker are then sent to the parser's `process()` method for processing in the usual way.

**Continuous Load**

The `ContinuousUDSource`, `ContinuousUDFilter`, and `ContinuousUDParser` classes allow you to write and process data as needed instead of having to iterate through the data.

Each class includes the following functions:

- `initialize()` - Invoked before `run()`. You can optionally override this function to perform setup and initialization.
- `run()` - Processes the data.
- `deinitialize()` - Invoked after `run()` has returned. You can optionally override this function to perform tear-down and destruction.

Do not override the `setup()`, `process()`, and `destroy()` functions that are inherited from parent classes.

You can use the `yield()` function to yield control back to the server during idle or busy loops so the server can check for status changes or query cancellations.

These three classes use associated `ContinuousReader` and `ContinuousWriter` classes to read input data and write output data.

**C++ API**

This section provides APIs and examples for the C++ API for UDLs.

For information on setting up a C++ development environment and compiling and packaging libraries, see [Developing with the C++ SDK](#).

**Requirements for C++ UDLs**

C++ UDLs:

- Can run in Fenced Mode. Vertica enables fenced mode by default when you create a `source`, `filter`, or `parser` function, in Vertica unless you explicitly state otherwise.
Must not permit an exception to be passed back to Vertica. Doing so could lead to issues such as memory leaks caused by the memory allocated by the exception never being freed. Your UDL should always contain a top-level try-catch block to catch any stray exceptions caused by your code or libraries that your code calls.

Must properly free any resources that the UDL function allocates. Even a single byte of allocated memory that is not freed can become an issue in a UDL that is called over millions of rows. Instead of allocating memory directly, your function should use the memory allocation macros in the Vertica SDK. See Allocating Resources for UDxs for details.

The header files that define the majority of classes and methods are VerticaUDx.h and VerticaUDl.h. These header files, along with the main Vertica.h header file, are available in /opt/vertica/sdk/include.

**Source Classes**

See User-Defined Source for general information about implementing the UDSource and SourceFactory classes. This section describes information that is specific to the C++ API.

**UDSource API**

The API provides the following methods for extension by subclasses:

```cpp
virtual void setup (ServerInterface &srvInterface);
virtual StreamState process (ServerInterface &srvInterface,
                               DataBuffer &output)=0;
virtual void destroy (ServerInterface &srvInterface);
virtual vint getSize ();
virtual std::string getUri ();
```

**ContinuousUDSource API**

The ContinuousUDSource class extends UDSource and adds the following methods for extension by subclasses:

```cpp
virtual void initialize (ServerInterface &srvInterface);
virtual void run ();
virtual void deinitialize (ServerInterface &srvInterface);
```
SourceFactory API

The API provides the following methods for extension by subclasses:

```cpp
virtual void plan (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt);
// must implement exactly one of prepareUDSources() or prepareUDSourcesExecutor()
virtual std::vector< UDSource * > prepareUDSources (ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtxt);
virtual std::vector< UDSource * > prepareUDSourcesExecutor (ServerInterface &srvInterface, ExecutorPlanContext &planCtxt);
virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes);
virtual bool isSourceApportionable ();
ssize_t getDesiredThreads (ServerInterface &srvInterface, ExecutorPlanContext &planContext);
```

After creating your SourceFactory, you must register it with the RegisterFactory macro.

Filter Classes

This section describes information that is specific to the C++ API. See User-Defined Filter for general information about implementing the UDFilter and FilterFactory classes.

UDFilter API

The API provides the following methods for extension by subclasses:

```cpp
virtual void setup (ServerInterface & srvInterface);
virtual StreamState process (ServerInterface & srvInterface, DataBuffer &input, InputState input_state, DataBuffer &output)=0;
virtual void destroy (ServerInterface &srvInterface);
```

ContinuousUDFilter API

The ContinuousUDFilter class extends UDFilter and adds the following methods for extension by subclasses:

```cpp
virtual void initialize (ServerInterface &srvInterface);
virtual void run ();
```
virtual void deinitialize (ServerInterface &srvInterface);

FilterFactory API

The API provides the following methods for extension by subclasses:

virtual void plan (ServerInterface &srvInterface, PlanContext &planCtx);
virtual UDFilter * prepare (ServerInterface &srvInterface, PlanContext &planCtx)=0;
virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes);

After creating your FilterFactory, you must register it with the RegisterFactory macro.

Parser Classes

This section describes information that is specific to the C++ API. See User-Defined Parser for general information about implementing the UDParser and ParserFactory classes.

UDParser API

The API provides the following methods for extension by subclasses:

virtual void setup (ServerInterface &srvInterface, SizedColumnTypes &returnType);
virtual StreamState process (ServerInterface &srvInterface, DataBuffer &input, InputState input_state)=0;
virtual void destroy (ServerInterface &srvInterface, SizedColumnTypes &returnType);
virtual RejectedRecord getRejectedRecord ();

Rejecting Records

The process() method might need to reject input. To reject data, you must do two things:

- Create a getRejectedRecord() method on your UDParser subclass that returns an object of type Vertica::RejectedRecord. This record contains the data that you want to reject, a string describing the reason for rejection, the size of the data, and the terminator string. See RejectedRecord in VerticaUDI.h or the SDK documentation for
details.

- Reject a row by returning REJECT from process(). Vertica then calls
  getRejectedRecord() to process the rejected record before the next call to process()

You can fulfill these requirements by including code in your parser class such as:

```cpp
Vertica::RejectedRecord myRejRec;
Vertica::RejectedRecord getRejectedRecord() {
    return myRejRec;
}
```

In your process() method, add code such as:

```cpp
if (some rejection condition) {
    RejectedRecord rr("Bad Record!", "foo data", 8, "\n");
    myRejRec = rr;
    return Vertica::REJECT;
}
```

ContinuousUDParser API

The ContinuousUDParser class extends UDParser and adds the following methods for extension by subclasses:

```cpp
virtual void initialize (ServerInterface &srvInterface);
virtual void run ();
virtual void deinitialize (ServerInterface &srvInterface);
```

UDChunker API

The UDChunker API provides the following methods for extension by subclasses:

```cpp
virtual void setup (ServerInterface &srvInterface,
                    SizedColumnTypes &returnType);
virtual StreamState alignPortion (ServerInterface &srvInterface,
                                   DataBuffer &input, InputState state);
virtual StreamState process (ServerInterface &srvInterface,
                           DataBuffer &input, InputState input_state=0);
virtual void destroy (ServerInterface &srvInterface,
                     SizedColumnTypes &returnType);
```
ParserFactory API

The API provides the following methods for extension by subclasses:

```c++
virtual void plan (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt);

virtual UDParse * prepare (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader,
                                 PlanContext &planCtxt, const SizedColumnTypes &returnType)=0;

virtual void getParameterType (ServerInterface &srvInterface, SizedColumnTypes &parameterTypes);

virtual void getParserReturnType (ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader,
                                                  PlanContext &planCtxt, const SizedColumnTypes &argTypes,
                                                  SizedColumnTypes &returnType);

virtual bool isParserApportionable (){
  // C++ API only:
  virtual bool isChunkerApportionable (ServerInterface &srvInterface);

  virtual UDChunker * prepareChunker (ServerInterface &srvInterface, PerColumnParamReader
                                                &perColumnParamReader,
                                              PlanContext &planCtxt, const SizedColumnTypes &returnType);

  if you are using Apportioned Load to divide a single input into multiple load streams,
  implement isParserApportionable() and/or isChunkerApportionable() and return true. Returning true
  from these methods does not guarantee that Vertica will apportion the load. However, returning false
  from both indicates that it will not try to do so.

  If you are using Cooperative Parse, implement prepareChunker() and return an instance of
  your UDChunker subclass. Cooperative parse is supported only for the C++ API.

  Vertica calls the prepareChunker() method only for unfenced functions. This method is not
  available when you use the function in fenced mode.

  If you want your chunker to be available for apportioned load, implement
  isChunkerApportionable() and return true.

  After creating your ParserFactory, you must register it with the RegisterFactory macro.

Source Example: CurlSource

The CurlSource example allows you to use cURL to open and read in a file over HTTP. The
example provided is part of:
/opt/vertica/sdk/examples/SourceFunctions/cURL.cpp.
Parser Implementation

This example uses the helper library available in /opt/vertica/sdk/examples/HelperLibraries/.

CurlSource loads the data in chunks. If the parser encounters an EndOfFile marker, then the process() method returns DONE. Otherwise, the method returns OUTPUT_NEEDED and processes another chunk of data. The functions included in the helper library (such as url_fread() and url_fopen()) are based on examples that come with the libcurl library. For an example, see http://curl.haxx.se/libcurl/c/fopen.html.

The setup() function opens a file handle and the destroy() function closes it. Both use functions from the helper library.

```cpp
class CurlSource : public UDSource {private:
   URL_FILE *handle;
   std::string url;
   virtual StreamState process(ServerInterface &srvInterface, DataBuffer &output) {
      output.offset = url_fread(output.buf, 1, output.size, handle);
      return urlfeof(handle) ? DONE : OUTPUT_NEEDED;
   }
public:
   CurlSource(std::string url) : url(url) {}
   void setup(ServerInterface &srvInterface) {
      handle = url_fopen(url.c_str(), "r");
   }
   void destroy(ServerInterface &srvInterface) {
      url_fclose(handle);
   }
};
```

Factory Implementation

CurlSourceFactory produces CurlSource instances.

```cpp
class CurlSourceFactory : public SourceFactory {public:
   virtual void plan(ServerInterface &srvInterface, NodeSpecifyingPlanContext &planCtx) {
      std::vector<std::string> args = srvInterface.getParamReader().getParamNames();
      /* Check parameters */
      if (args.size() != 1 || find(args.begin(), args.end(), "url") == args.end()) {
         vt_report_error(0, "You must provide a single URL.");
      }
      /* Populate planData */
      planCtx.getWriter().getStringRef("url").copy(  
         srvInterface.getParamReader().getStringRef("url"));
      /* Assign Nodes */
      std::vector<std::string> executionNodes = planCtx.getClusterNodes();
      while (executionNodes.size() > 1) executionNodes.pop_back();
      // Only run on the first node in the list.
   }
};
```
planCtx.setTargetNodes(executionNodes);
}
virtual std::vector<UDSource*> prepareUDSources(ServerInterface &srvInterface,
NodeSpecifyingPlanContext &planCtxt) {
std::vector<UDSource*> retVal;
retVal.push_back(vt_createFuncObj(srvInterface.allocator, CurlSource,
planCtxt.getReader().getStringRef("url").str()));
return retVal;
}
virtual void getParameterType(ServerInterface &srvInterface,
SizedColumnTypes &parameterTypes) {
parameterTypes.addVarchar(65000, "url");
};
RegisterFactory(CurlSourceFactory);

Source Example: Concurrent Load

The FilePortionSource example demonstrates the use of concurrent load. This example is a refinement of the FileSource example. Each input file is divided into portions and distributed to FilePortionSource instances. The source accepts a list of offsets at which to break the input into portions; if offsets are not provided, the source divides the input dynamically.

Concurrent load is handled in the factory, so this discussion focuses on FilePortionSourceFactory. The full code for the example is located in /opt/vertica/sdk/examples/ApportionLoadFunctions. The distribution also includes a Java version of this example.

Loading and Using the Example

Load and use the FilePortionSource example as follows.

```sql
=> CREATE LIBRARY FilePortionLib AS '/home/dbadmin/FP.so';
=> CREATE SOURCE FilePortionSource AS LANGUAGE 'C++'
  -> NAME 'FilePortionSourceFactory' LIBRARY FilePortionLib;
=> COPY t WITH SOURCE FilePortionSource(file='g1/*.dat', nodes='initiator,e0,e1', offsets = '0,380000,820000');
=> COPY t WITH SOURCE FilePortionSource(file='g2/*.dat', nodes='e0,e1,e2', local_min_portion_size = 2097152);
```
Implementation

Concurrent load affects the source factory in two places, `getDesiredThreads()` and `prepareUDSourcesExecutor()`.

**getDesiredThreads()**

The `getDesiredThreads()` member function determines the number of threads to request. Vertica calls this member function on each executor node before calling `prepareUDSourcesExecutor()`.

The function begins by breaking an input file path, which might be a glob, into individual paths. This discussion omits those details. If apportioned load is not being used, then the function allocates one source per file.

```cpp
virtual ssize_t getDesiredThreads(ServerInterface &srvInterface,
   ExecutorPlanContext &planCtxt) {
   const std::string filename = srvInterface.getParamReader().getStringRef("file").str();

   std::vector<std::string> paths;
   // expand the glob - at least one thread per source.
   ...

   // figure out how to assign files to sources
   const std::string nodeName = srvInterface.getCurrentNodeName();
   const size_t nodeId = planCtxt.getWriter().getIntRef(nodeName);
   const size_t numNodes = planCtxt.getTargetNodes().size();

   if (!planCtxt.canApportionSource()) {
      /* no apportioning, so the number of files is the final number of sources */
      std::vector<std::string> *expanded =
          vt_createFuncObject<std::vector<std::string>> >(srvInterface.allocator, paths);
      /* save expanded paths so we don't have to compute expansion again */
      planCtxt.getWriter().setPointer("expanded", expanded);
      return expanded->size();
   }
   // ...

   else if (srvInterface.getParamReader().containsParameter("offsets")) {

      // if the offsets are specified, then we will have a fixed number of portions per file.
```
// Round-robin assign offsets to nodes.  
// ...  

/* Construct the portions that this node will actually handle, 
* This isn't changing (since the offset assignments are fixed),
* so we'll build the Portion objects now and make them available 
* to prepareUDSourcesExecutor() by putting them in the ExecutorContext.  
* 
* We don't know the size of the last portion, since it depends on the file 
* size. Rather than figure it out here we will indicate it with -1 and 
* defer that to prepareUDSourcesExecutor(). */
std::vector<Portion>*portions =
    vt_createFuncObject<std::vector<Portion>>(srvInterface.allocator);

for (std::vector<size_t>::const_iterator offset = offsets.begin();
    offset != offsets.end(); ++offset) {
    Portion p(*offset);
    p.is_first_portion = (offset == offsets.begin());
    p.size = (offset + 1 == offsets.end()) ? -1 : (*offset + 1) - *offset);

    if (((offset - offsets.begin()) % numNodes == nodeId) {
        portions->push_back(p);
        srvInterface.log("FilePortionSource: assigning portion %ld: [offset = %lld, size =
                        %lld]",
                        offset - offsets.begin(), p.offset, p.size);
    }
}

The function now has all the portions and thus the number of portions:

    planCtx.getWriter().setPointer("portions", portions);

    /* total number of threads we want is the number of portions per file, which is fixed */
    return portions->size() * expanded->size();
} // end of "offsets" parameter

If offsets were not provided, the function divides the file into portions dynamically, one portion 
per thread. This discussion omits the details of this computation. There is no point in 
requesting more threads than are available, so the function calls getMaxAllowedThreads() 
on the PlanContext (an argument to the function) to set an upper bound:

    if (portions->size() >= planCtx.getMaxAllowedThreads()) {
        return paths.size();
    }

See the full example for the details of how this function divides the file into portions.

This function uses the vt_createFuncObject template to create objects. Vertica calls the 
destructors of returned objects created using this macro, but it does not call destructors for 
other objects like vectors. You must call these destructors yourself to avoid memory leaks. In 
this example, these calls are made in prepareUDSourcesExecutor().
prepareUDSourcesExecutor()

The prepareUDSourcesExecutor() member function, like getDesiredThreads(), has separate blocks of code depending on whether offsets are provided. In both cases, the function breaks input into portions and creates UDSource instances for them.

If the function is called with offsets, prepareUDSourcesExecutor() calls prepareCustomizedPortions(). This function follows.

```c++
/* prepare portions as determined via the "offsets" parameter */
void prepareCustomizedPortions(ServerInterface &srvInterface,
                                ExecutorPlanContext &planCtxt,
                                std::vector<UDSource *> &sources,
                                const std::vector<std::string> &expandedPaths,
                                std::vector<Portion> &portions) {
  for (std::vector<Portion>::const_iterator filename = expandedPaths.begin();
       filename != expandedPaths.end(); ++filename) {
    /*
     * the "portions" vector contains the portions which were generated in
     * "getDesiredThreads"
     */
    const size_t fileSize = getFileSize(*filename);
    for (std::vector<Portion>::const_iterator portion = portions.begin();
         portion != portions.end(); ++portion) {
      Portion fportion(*portion);
      if (fportion.size == -1) {
        /* as described above, this means from the offset to the end */
        fportion.size = fileSize - portion->offset;
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,
                                                                  *filename, fportion));
      } else if (fportion.size > 0) {
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,
                                                                  *filename, fportion));
      }
    }
  }
}
```

If prepareUDSourcesExecutor() is called without offsets, then it must decide how many portions to create.

The base case is to use one portion per source. However, if extra threads are available, the function divides the input into more portions so that a source can process them concurrently. Then prepareUDSourcesExecutor() calls prepareGeneratedPortions() to create the portions. This function begins by calling getLoadConcurrency() on the plan context to find out how many threads are available.

```c++
void prepareGeneratedPortions(ServerInterface &srvInterface,
                               ExecutorPlanContext &planCtxt,
                               std::vector<UDSource *> &sources,
                               std::vector<Portion> &portions) {
  const size_t totalFilesize = getFileSize(...);
  if (getLoadConcurrency() > 1) {
    /* divide the input into portions */
    ... (divide input into portions)
  }
  prepareCustomizedPortions(srvInterface, planCtxt, sources, expandedPaths, portions);
}
```
std::map<std::string, Portion> initialPortions) {
if ((ssize_t) initialPortions.size() >= planCtxt.getLoadConcurrency()) {
    /* all threads will be used, don’t bother splitting into portions */
    for (std::map<std::string, Portion>::const_iterator file = initialPortions.begin();
        file != initialPortions.end(); ++file) {
        sources.push_back(vt_createFuncObject<FilePortionSource>(srvInterface.allocator,
            file->first, file->second));
    } // for
    return;
} // if

// Now we can split files to take advantage of potentially-unused threads.
// First sort by size (descending), then we will split the largest first.
// details elided...
}

For More Information
See the source code for the full implementation of this example.

Filter Example: Converting Encoding

The following example shows how you can convert encoding for a file from one type to
another by converting UTF-16 encoded data to UTF-8. You can find this example in the SDK at
/opt/vertica/sdk/examples/FilterFunctions/IConverter.cpp.

Filter Implementation

class IConverter : public UDFilter{
private:
    std::string fromEncoding, toEncoding;
    iconv_t cd; // the conversion descriptor opened
    uint converted; // how many characters have been converted
protected:
    virtual StreamState process(ServerInterface &srvInterface, DataBuffer &input,
        InputState input_state, DataBuffer &output) {
        char *input_buf = (char *)input.buf + input.offset;
        char *output_buf = (char *)output.buf + output.offset;
        size_t inBytesLeft = input.size - input.offset, outBytesLeft = output.size - output.offset;
        // end of input
        if (input_state == END_OF_FILE && inBytesLeft == 0) {
            // Gnu libc iconv doc says, it is good practice to finalize the
            // outbuffer for stateful encodings (by calling with null inbuffer).
Vertica Analytic Database (9.0.x)
Factory Implementation

class IConverterFactory : public FilterFactory
{
    public:
        virtual void plan(ServerInterface &srvInterface,
            PlanContext &planCtxt) {
            std::vector<std::string> args = srvInterface.getParamReader().getParamNames();
            /* Check parameters */
            if (!(args.size() == 0 || (args.size() == 1 && find(args.begin(), args.end(), "from_encoding") != args.end())
                && find(args.begin(), args.end(), "from_encoding") != args.end())
                && find(args.begin(), args.end(), "to_encoding") != args.end())) {
                vt_report_error(0, "Invalid arguments. Must specify either no arguments, or "
                    "'from_encoding' alone, or 'from_encoding' and 'to_encoding'.");
            } /* Populate planData */
            // By default, we do UTF16->UTF8, and x->UTF8
            VString from_encoding = planCtxt.getWriter().getStringRef("from_encoding");
            VString to_encoding = planCtxt.getWriter().getStringRef("to_encoding");
            from_encoding.copy("UTF-16");
            to_encoding.copy("UTF-8");
            if (args.size() == 2) {
                from_encoding.copy(srvInterface.getParamReader().getStringRef("from_encoding"));
                to_encoding.copy(srvInterface.getParamReader().getStringRef("to_encoding"));
            } else if (args.size() == 1) {
                from_encoding.copy(srvInterface.getParamReader().getStringRef("from_encoding"));
            } if (!from_encoding.length()) {
                vt_report_error(0, "The empty string is not a valid from_encoding value");
            } if (!to_encoding.length()) {
                vt_report_error(0, "The empty string is not a valid to_encoding value");
            }
        } virtual UDFilter* prepare(ServerInterface &srvInterface,
            PlanContext &planCtxt) {
            return vt_createFuncObj(srvInterface.getAllocator(), IConverter,
                planCtxt.getReader().getStringRef("from_encoding").str(),
                planCtxt.getReader().getStringRef("to_encoding").str());
        } virtual void getParameterType(ServerInterface &srvInterface,
            SizedColumnTypes &parameterTypes) {
            parameterTypes.addVarchar(32, "from_encoding");
            parameterTypes.addVarchar(32, "to_encoding");
        }};
    RegisterFactory(IConverterFactory);
Parser Example: BasicIntegerParser

The BasicIntegerParser example parses a string of integers separated by non-numeric characters. For a version of this parser that uses continuous load, see Parser Example: ContinuousIntegerParser.

Loading and Using the Example

Load and use the BasicIntegerParser example as follows.

```sql
=> CREATE LIBRARY BasicIntegerParserLib AS '/home/dbadmin/BIP.so';
=> CREATE PARSER BasicIntegerParser AS LANGUAGE 'C++' NAME 'BasicIntegerParserFactory' LIBRARY BasicIntegerParserLib;
=> CREATE TABLE t (i integer);
=> COPY t FROM stdin WITH PARSER BasicIntegerParser();
0
1
2
3
4
5
.
```

Implementation

The BasicIntegerParser class implements only the `process()` method from the API. (It also implements a helper method for type conversion.) This method processes each line of input, looking for numbers on each line. When it advances to a new line it moves the `input.offset` marker and checks the input state. It then writes the output.

```cpp
virtual StreamState process(ServerInterface &srvInterface, DataBuffer &input, InputState input_state) {
    // WARNING: This implementation is not trying for efficiency.
    // It is trying for simplicity, for demonstration purposes.

    size_t start = input.offset;
    const size_t end = input.size;

    do {
        bool found_newline = false;
        size_t numEnd = start;
        for (; numEnd < end; numEnd++) {
            if (input.buf[numEnd] < '0' || input.buf[numEnd] > '9') {
                found_newline = true;
                break;
            }
        }
```
if (!found_newline) {
    input.offset = start;
    if (input_state == END_OF_FILE) {
        // If we're at end-of-file, emit the last integer (if any) and return DONE.
        if (start != end) {
            writer->setInt(0, strToInt(input.buf + start, input.buf + numEnd));
            writer->next();
        } // Otherwise, we need more data.
        return INPUT_NEEDED;
    } else {
    
    }
}

writer->setInt(0, strToInt(input.buf + start, input.buf + numEnd));
writer->next();

start = numEnd + 1;
} while (true);

In the factory, the `plan()` method is a no-op; there are no parameters to check. The `prepare()` method instantiates the parser using the macro provided by the SDK:

```cpp
virtual UDParser* prepare(ServerInterface &srvInterface, 
PerColumnParamReader &perColumnParamReader, 
PlanContext &planCtx, 
const SizedColumnTypes &returnType) {

    return vt_createFuncObject<BasicIntegerParser>(srvInterface.allocator);
}
```

The `getParserReturnType()` method declares the single output:

```cpp
virtual void getParserReturnType(ServerInterface &srvInterface, 
PerColumnParamReader &perColumnParamReader, 
PlanContext &planCtx, 
const SizedColumnTypes &argTypes, 
SizedColumnTypes &returnType) {
    // We only and always have a single integer column
    returnType.addInt(argTypes.getColumnType(0));
}
```

As for all UDxs written in C++, the example ends by registering its factory:

```cpp
RegisterFactory(BasicIntegerParserFactory);
```
Parser Example: ContinuousIntegerParser

The ContinuousIntegerParser example is a variation of BasicIntegerParser. Both examples parse integers from input strings. ContinuousIntegerParser uses Continuous Load to read data.

Loading and Using the Example

Load the ContinuousIntegerParser example as follows.

```sql
=> CREATE LIBRARY ContinuousIntegerParserLib AS '/home/dbadmin/CIP.so';

=> CREATE PARSER ContinuousIntegerParser AS
   LANGUAGE 'C++' NAME 'ContinuousIntegerParserFactory'
   LIBRARY ContinuousIntegerParserLib;
```

Use it in the same way that you use BasicIntegerParser. See Parser Example: BasicIntegerParser.

Implementation

ContinuousIntegerParser is a subclass of ContinuousUDParser. Subclasses of ContinuousUDParser place the processing logic in the run() method.

```c++
virtual void run() {
    // This parser assumes a single-column input, and
    // a stream of ASCII integers split by non-numeric characters.
    size_t pos = 0;
    size_t reserved = cr.reserve(pos+1);
    while (!cr.isEof() || reserved == pos + 1) {
        while (reserved == pos + 1 && isdigit(*ptr(pos))) {
            pos++;
            reserved = cr.reserve(pos + 1);
        }
        std::string st(ptr(), pos);
        writer->setInt(0, strToInt(st));
        writer->next();
        while (reserved == pos + 1 && !isdigit(*ptr(pos))) {
            pos++;
            reserved = cr.reserve(pos + 1);
        }
        cr.seek(pos);
        pos = 0;
        reserved = cr.reserve(pos + 1);
    }
}
```
For a more complex example of a ContinuousUDParser, see ExampleDelimitedParser in the examples. (See Downloading and Running UDx Example Code.) ExampleDelimitedParser uses a chunker; see Chunker Example: Delimited Parser and Chunker.

Chunker Example: Delimited Parser and Chunker

The ExampleDelimitedUDChunker class divides an input at delimiter characters. You can use this chunker with any parser that understands delimited input. ExampleDelimitedParser is a ContinuousUDParser subclass that uses this chunker.

Loading and Using the Example

Load and use the example as follows.

```sql
=> CREATE LIBRARY ExampleDelimitedParserLib AS '/home/dbadmin/EDP.so';

=> CREATE PARSER ExampleDelimitedParser AS
   LANGUAGE 'C++' NAME 'DelimitedParserFrameworkExampleFactory'
   LIBRARY ExampleDelimitedParserLib;

=> COPY t FROM stdin WITH PARSER ExampleDelimitedParser();
```

Chunker Implementation

This chunker supports apportioned load. The alignPortion() method finds the beginning of the first complete record in the current portion and aligns the input buffer with it. The record terminator is passed as an argument and set in the constructor.

```cpp
StreamState ExampleDelimitedUDChunker::alignPortion(    
    ServerInterface &srvInterface,    
    DataBuffer &input, InputState state)    
{
```
The `process()` method has to account for chunks that span portion boundaries. If the previous call was at the end of a portion, the method set a flag. The code begins by checking for and handling that condition. The logic is similar to that of `alignPortion()`, so the example calls it to do part of the division.

```c++
StreamState ExampleDelimitedUDChunker::process(
    ServerInterface &srvInterface,
    DataBuffer &input,
    InputState input_state)
{
    const size_t termLen = 1;
    const char *terminator = &recordTerminator;

    if (pastPortion) {
        /*
         * Previous state was END_OF_PORTION, and the last chunk we will produce
         * extends beyond the portion we started with, into the next portion.
         * To be consistent with `alignPortion()`, that means finding the first
         * record boundary, and setting the chunk to be at that boundary.
         * Fortunately, this logic is identical to aligning the portion (with
         * some slight accounting for END_OF_FILE)!
         */
        const StreamState findLastTerminator = alignPortion(srvInterface, input);

        switch (findLastTerminator) {
            case DONE:
                return DONE;
            case INPUT_NEEDED:
                if (input_state == END_OF_FILE) {
                    /* there is no more input where we might find a record terminator */
                    input.offset = input.size;
                    return DONE;
                }

                return INPUT_NEEDED;
            default:
                VIAssert("Invalid return state from alignPortion()");
        }
    }
```
return findLast Terminator;
}

Now the method looks for the delimiter. If the input began at the end of a portion, it sets the flag.

size_t ret = input.offset, term_index = 0;
for (size_t index = input.offset; index < input.size; ++index) {
    const char c = input.buf[index];
    if (c == terminator[term_index]) {
        ++term_index;
        if (term_index == termLen) {
            ret = index + 1;
            term_index = 0;
        }
        continue;
    } else if (term_index > 0) {
        index -= term_index;
    }
    term_index = 0;
}

if (input_state == END_OF_PORTION) {
    /*
     * Regardless of whether or not a record was found, the next chunk will extend
     * into the next portion.
     */
    pastPortion = true;
}

Finally, process() moves the input offset and returns.

// if we were able to find some rows, move the offset to point at the start of the next
(potential) row, or end of block
if (ret > input.offset) {
    input.offset = ret;
    return CHUNK_ALIGNED;
}

if (input_state == END_OF_FILE) {
    input.offset = input.size;
    return DONE;
}

return INPUT_NEEDED;

Factory Implementation

The file ExampleDelimitedParser.cpp defines a factory that uses this UDChunker. The chunker supports apportioned load, so the factory implements isChunkerApportionable():
The `prepareChunker()` method creates the chunker:

```cpp
virtual UDChunker* prepareChunker(ServerInterface &srvInterface, PerColumnParamReader &perColumnParamReader, PlanContext &planCtxt, const SizedColumnType &returnType) {
    ParamReader params = srvInterface.getParamReader();
    if (params.containsParameter("disable_chunker") && params.getBoolRef("disable_chunker")) {
        return NULL;
    }
    std::string recordTerminator("\n");
    ParamReader args(srvInterface.getParamReader());
    if (args.containsParameter("record_terminator")) {
        recordTerminator = args.getStringRef("record_terminator").str();
    }
    return vt_createFuncObject<ExampleDelimitedUDChunker>(srvInterface.allo\nos, recordTerminator[0]);
}
```

### Java API

The Vertica Java SDK supports developing UDLs. The Java SDK enables you to create sources, filters, and parsers, but not chunkers.

For information on setting up a Java development environment and compiling and packaging libraries, see [Developing with the Java SDK](#).

### Source Classes

This section describes information that is specific to the Java API. See [User-Defined Source](#) for general information about implementing the UDSource and SourceFactory classes.
UDSource API

The API provides the following methods for extension by subclasses:

```java
public void setup (ServerInterface srvInterface) throws UdfException;
public abstract StreamState process (ServerInterface srvInterface, DataBuffer output) throws UdfException;
public void destroy (ServerInterface srvInterface) throws UdfException;
public Integer getSize ();
public String getUri ();
```

SourceFactory API

The API provides the following methods for extension by subclasses:

```java
public void plan (ServerInterface srvInterface, NodeSpecifyingPlanContext planCtxt) throws UdfException;
// must implement one overload of prepareUDSources()
public ArrayList<UDSource> prepareUDSources (ServerInterface srvInterface, NodeSpecifyingPlanContext planCtxt) throws UdfException;
public ArrayList<UDSource> prepareUDSources (ServerInterface srvInterface, ExecutorPlanContext planCtxt) throws UdfException;
public void getParameterType (ServerInterface srvInterface, SizedColumnTypes parameterTypes);
public boolean isSourceApportionable();
public int getDesiredThreads (ServerInterface srvInterface, ExecutorPlanContext planCtxt) throws UdfException;
```

Filter Classes

This section describes information that is specific to the Java API. See User-Defined Filter for general information about implementing the UDFilter and FilterFactory classes.

UDFilter API

The API provides the following methods for extension by subclasses:
public void setup (ServerInterface srvInterface) throws UdfException;
public abstract StreamState process (ServerInterface srvInterface, DataBuffer input,
        InputState input_state, DataBuffer output) throws UdfException;
public void destroy (ServerInterface srvInterface) throws UdfException;

FilterFactory API
The API provides the following methods for extension by subclasses:

public void plan (ServerInterface srvInterface, PlanContext planCtxt) throws UdfException;
public abstract UDFilter prepare (ServerInterface srvInterface, PlanContext planCtxt) throws UdfException;
public void getParameterType (ServerInterface srvInterface, SizedColumnTypes parameterTypes);

Parser Classes
This section describes information that is specific to the Java API. See User-Defined Parser for general information about implementing the UDParse and ParserFactory classes.

UDParser API
The API provides the following methods for extension by subclasses:

public void setup (ServerInterface srvInterface, SizedColumnTypes returnType) throws UdfException;
public abstract StreamState process (ServerInterface srvInterface, DataBuffer input, InputState input_state) throws UdfException, DestroyInvocation;
public void destroy (ServerInterface srvInterface, SizedColumnTypes returnType) throws UdfException;
public RejectedRecord getRejectedRecord () throws UdfException;

A UDParse uses a StreamWriter to write its output. StreamWriter provides methods for all the basic types, such as setBooleanValue (), setValue (), and so on. In the Java API this class also provides the setValue () method, which automatically sets the data type.
The methods described so far write single column values. StreamWriter also provides a method to write a complete row from a map. The `setRowFromMap()` method takes a map of column names and values and writes all the values into their corresponding columns. This method does not define new columns but instead writes values only to existing columns. The JsonParser example uses this method to write arbitrary JSON input. (See Parser Example: JSON Parser.)

Note: The `setRowFromMap()` method does not automatically advance the input to the next line; you must call `next()`. You can thus read a row and then override selected column values.

`setRowsFromMap()` also populates any VMap ("_raw_") column of Flex Tables (see Using Flex Tables) with the entire provided map. For most cases, `setRowsFromMap()` is the appropriate way to populate a Flex Table. However, you can also generate a VMap value into a specified column using `setVMap()`, similar to other `setValue()` methods.

The `setRowFromMap()` method automatically coerces the input values into the types defined for those columns using an associated `TypeCoercion`. In most cases, using the default implementation (StandardTypeCoercion) is appropriate.

TypeCoercion uses policies to govern its behavior. For example, the FAIL_INVALID_INPUT_VALUE policy means invalid input is treated as an error instead of using a null value. Errors are caught and handled as rejections (see "Rejecting Rows" in User-Defined Parser). Policies also govern whether input that is too long is truncated. Use the `setPolicy()` method on the parser's `TypeCoercion` to set policies. See the API documentation for supported values.

You might need to customize type coercion beyond setting these policies. To do so, subclass one of the provided implementations of TypeCoercion and override the `asType()` methods. Such customization could be necessary if your parser reads objects that come from a third-party library. A parser handling geo-coordinates, for example, might override `asLong` to translate inputs like "40.4397N" into numbers. See the Vertica API documentation for a list of implementations.

ContinuousUDParser API

The ContinuousUDParser class extends UDPParser and adds the following methods for extension by subclasses:

```java
void initialize (ServerInterface srvInterface, SizedColumnTypes returnType);
abstract void run () throws UdfException;
void deinitialize (ServerInterface srvInterface, SizedColumnTypes returnType);
```

See the API documentation for additional utility methods.
ParserFactory API

The API provides the following methods for extension by subclasses:

```java
public void plan(ServerInterface srvInterface, PerColumnParamReader perColumnParamReader, PlanContext planCtxt)
    throws UdfException;

public abstract UDParse prepare(ServerInterface srvInterface, PerColumnParamReader
    perColumnParamReader, PlanContext planCtxt, SizedColumnTypes returnType)
    throws UdfException;

public void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes);

public void getParserReturnType(ServerInterface srvInterface, PerColumnParamReader
    perColumnParamReader, PlanContext planCtxt, SizedColumnTypes argTypes, SizedColumnTypes returnType)
    throws UdfException;
```

Source Example: FileSource

The example shown in this section is a simple UDL Source function named FileSource, This function loads data from files stored on the host's file system (similar to the standard COPY statement). To call FileSource, you must supply a parameter named file that contains the absolute path to one or more files on the host file system. You can specify multiple files as a comma-separated list.

The FileSource function also accepts an optional parameter, named nodes, that indicates which nodes should load the files. If you do not supply this parameter, the function defaults to loading data on the initiator node only. Because this example is simple, the nodes load only the files from their own file systems. Any files in the file parameter must exist on all of the hosts in the nodes parameter. The FileSource UDSource attempts to load all of the files in the file parameter on all of the hosts in the nodes parameter.

Generating Files

You can use the following Python script to generate files and distribute them to hosts in your Vertica cluster. With these files, you can experiment with the example UDSource function. Running the function requires passwordless-SSH logins to copy the files to the other hosts. Therefore, you must run the script using the database administrator account on one of your database hosts.
#!/usr/bin/python
# Save this file as UDLDataGen.py
import string
import random
import sys
import os

# Read in the dictionary file to provide random words. Assumes the words
# file is located in /usr/share/dict/words
wordFile = open('/usr/share/dict/words')
wordDict = []
for line in wordFile:
    if len(line) > 6:
        wordDict.append(line.strip())

MAXSTR = 4 # Maximum number of words to concatenate
NUMROWS = 1000 # Number of rows of data to generate
FILEPATH = '/tmp/UDLdata.txt' # Final filename to use for UDL source
TMPFILE = '/tmp/UDLtemp.txt' # Temporary filename.

# Generate a random string by concatenating several words together. Max
# number of words set by MAXSTR
def randomWords():
    words = [random.choice(wordDict) for n in xrange(random.randint(1, MAXSTR))]
    sentence = " ".join(words)
    return sentence

# Create a temporary data file that will be moved to a node. Number of
# rows for the file is set by NUMROWS. Adds the name of the node which will
# get the file, to show which node loaded the data.
def generateFile(node):
    outFile = open(TMPFILE, 'w')
    for line in xrange(NUMROWS):
        outFile.write('{{0}}{{1}}{{2}}\n'.format(line,randomWords(),node))
    outFile.close()

# Copy the temporary file to a node. Only works if passwordless SSH login
# is enabled, which it is for the database administrator account on
# Vertica hosts.
def copyFile(fileName,node):
    os.system('scp "%s" "%s:%s"' % (TMPFILE, node, fileName) )

# Loop through the comma-separated list of nodes given in the first
# parameter, creating and copying data files whose full comma-separated
# paths are passed in the second parameter
for node in [x.strip() for x in sys.argv[1].split(',')]:
    for fileName in [y.strip() for y in sys.argv[2].split(',')]:
        print "generating file", fileName, "for", node
        generateFile(node)
        print "Copying file to",node
        copyFile(fileName,node)

You call this script by giving it a comma-separated list of hosts to receive the files and a
comma-separated list of absolute paths of files to generate. For example:

python UDLDataGen.py v_vmart_node0001,v_vmart_node0002,v_vmart_node0003
/tmp/UDLdata01.txt,/tmp/UDLdata02.txt,\
UDLdata03.txt
This script generates files that contain a thousand rows of columns delimited with the pipe character (|). These columns contain an index value, a set of random words, and the node for which the file was generated, as shown in the following output sample:

<table>
<thead>
<tr>
<th>0</th>
<th>megabits</th>
<th>embanks</th>
<th>v_vmart_node0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>neatly</td>
<td>v_vmart_node0001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>self-precipitation</td>
<td>v_vmart_node0001</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>antihistamine</td>
<td>scalados</td>
<td>Vatter</td>
</tr>
</tbody>
</table>

Loading and Using the Example

Load and use the FileSource UDSource as follows:

```
=> -- Load library and create the source function
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaUDLlib.jar'
=> LANGUAGE 'JAVA';
CREATE LIBRARY
=> CREATE SOURCE File as LANGUAGE 'JAVA' NAME
=> 'com.mycompany.UDL.FileSourceFactory' LIBRARY JavaLib;
CREATE SOURCE FUNCTION
=> -- Create a table to hold the data loaded from files
=> CREATE TABLE t (i integer, text VARCHAR, node VARCHAR);
CREATE TABLE
=> -- Copy a single file from the currently host using the FileSource
=> COPY t SOURCE File(file="/tmp/UDLdata01.txt");
Rows Loaded
----------
1000
(1 row)

=> -- See some of what got loaded.
=> SELECT * FROM t WHERE i < 5 ORDER BY i;

<table>
<thead>
<tr>
<th>i</th>
<th>text</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>megabits</td>
<td>embanks</td>
</tr>
<tr>
<td>1</td>
<td>neatly</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>2</td>
<td>self-precipitation</td>
<td>v_vmart_node0001</td>
</tr>
<tr>
<td>3</td>
<td>antihistamine</td>
<td>scalados</td>
</tr>
<tr>
<td>4</td>
<td>fate-menaced</td>
<td>toilworn</td>
</tr>
</tbody>
</table>

(5 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> -- Now load a file from three hosts. All of these hosts must have a file
=> -- named /tmp/UDLdata01.txt, each with different data
=> COPY t SOURCE File(file="/tmp/UDLdata01.txt",
=> nodes='v_vmart_node0001,v_vmart_node0002,v_vmart_node0003');
Rows Loaded
----------
3000
(1 row)

=> -- Now see what has been loaded
=> SELECT * FROM t WHERE i < 5 ORDER BY i,node;
```
### Extending Vertica (9.0.x = 10.0.x)

#### Implementation

When implementing a new source, such as a text file, to load into Vertica, the table setup code typically looks like the following:

```sql
TRUNCATE TABLE t;
COPY t SOURCE File(file='/tmp/UDLdata01.txt,/tmp/UDLdata02.txt,/tmp/UDLdata03.txt')

SELECT * FROM t WHERE i = 0 ORDER BY node;  
```

This example demonstrates how to load data from a non-textual file into Vertica. The `COPY` command is used to load data from a file, and the `SELECT` command is used to retrieve a subset of the loaded data. The `LOAD` and `COPY` commands in Vertica are used to load data into a table. They are similar, but there are some differences in how they work.

### Parser Implementation

The following code shows the source of the `FileSource` class that reads a file from the host file system. The constructor, which is called by `FileSourceFactory.prepareUDSources()`, gets the absolute path for the file containing the data to be read. The `setup()` method opens the file and the `destroy()` method closes it. The `process()` method reads from the file into a buffer provided by the instance of the `DataBuffer` class passed to it as a parameter. If the read operation filled the output buffer, it returns `OUTPUT_NEEDED`. This value tells Vertica to call the method again after the next stage of the load has processed the output buffer. If the read did not fill the output buffer, then `process()` returns `DONE` to indicate it has finished processing the data source.
package com.mycompany.UDL;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.RandomAccessFile;
import com.vertica.sdk.DataBuffer;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.State.StreamState;
import com.vertica.sdk.UDSource;
import com.vertica.sdk.UdfException;

public class FileSource extends UDSource {

    private String filename; // The file for this UDSource to read
    private RandomAccessFile reader; // handle to read from file

    // The constructor just stores the absolute filename of the file it will
    // read.
    public FileSource(String filename) {
        super();
        this.filename = filename;
    }

    // Called before Vertica starts requesting data from the data source.
    // In this case, setup needs to open the file and save to the reader
    // property.
    @Override
    public void setup(ServerInterface srvInterface) throws UdfException{
        try {
            reader = new RandomAccessFile(new File(filename), "r");
        } catch (FileNotFoundException e) {
            // In case of any error, throw a UdfException. This will terminate
            // the data load.
            String msg = e.getMessage();
            throw new UdfException(0, msg);
        }
    }

    // Called after data has been loaded. In this case, close the file handle.
    @Override
    public void destroy(ServerInterface srvInterface) throws UdfException {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                String msg = e.getMessage();
                throw new UdfException(0, msg);
            }
        }
    }

    // Read up to the size of the buffer provided in the DataBuffer.buf
    @Override
    public StreamState process(ServerInterface srvInterface, DataBuffer output)
        throws UdfException {
        // Read up to the size of the buffer provided in the DataBuffer.buf
    }
}
Factory Implementation

The following code is a modified version of the example Java UDsource function provided in the Java UDX support package. You can find the full example in /opt/vertica/sdk/examples/JavaUDx/UDLFuctions/com/vertica/JavaLibs/FileSourceFactory.java. Its override of the plan() method verifies that the user supplied the required file parameter. If the user also supplied the optional nodes parameter, this method verifies that the nodes exist in the Vertica cluster. If there is a problem with either parameter, the method throws an exception to return an error to the user. If there are no issues with the parameters, the plan() method stores their values in the plan context object.

```java
package com.mycompany.UDL;
import java.util.ArrayList;
import java.util.Vector;
import com.vertica.sdk.NodeSpecifyingPlanContext;
import com.vertica.sdk.ParamReader;
import com.vertica.sdk.ParamWriter;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.SourceFactory;
import com.vertica.sdk.UDSource;
import com.vertica.sdk.UdfException;

public class FileSourceFactory extends SourceFactory {

```
Called once on the initiator host to do initial setup. Checks parameters and chooses which nodes will do the work.

```java
@Override
public void plan(ServerInterface srvInterface,
                  NodeSpecifyingPlanContext planCtxt) throws UdfException {
    String nodes; // stores the list of nodes that will load data
    // Get copy of the parameters the user supplied to the UDSource
    // function call.
    ParamReader args = srvInterface.getParamReader();
    // A list of nodes that will perform work. This gets saved as part
    // of the plan context.
    ArrayList<String> executionNodes = new ArrayList<String>();
    // First, ensure the user supplied the file parameter
    if (!args.containsParameter("file")) {
        // Without a file parameter, we cannot continue. Throw an
        // exception that will be caught by the Java UDx framework.
        throw new UdfException(0, "You must supply a file parameter");
    }
    // If the user specified nodes to read the file, parse the
    // comma-separated list and save. Otherwise, assume just the
    // Initiator node has the file to read.
    if (args.containsParameter("nodes")) {
        nodes = args.getString("nodes");
        // Get list of nodes in cluster, to ensure that the node the
        // user specified actually exists. The list of nodes is available
        // from the planCtxt (plan context) object,
        ArrayList<String> clusterNodes = planCtxt.getClusterNodes();
        // Parse the string parameter "nodes" which
        // is a comma-separated list of node names.
        String[] nodeNames = nodes.split(",");
        for (int i = 0; i < nodeNames.length; i++){
            // See if the node the user gave us actually exists
            if (clusterNodes.contains(nodeNames[i])) {
                // Node exists. Add it to list of nodes.
                executionNodes.add(nodeNames[i]);
            } else {
                // User supplied node that doesn't exist. Throw an
                // exception so the user is notified.
                String msg = String.format("Specified node \%s\ but no'' +
                                          " node by that name is available. Available nodes " +
                                          "are \%s\"."
                                         , nodeNames[i], clusterNodes.toString());
                throw new UdfException(0, msg);
            }
        }
    } else {
        // User did not supply a list of node names. Assume the initiator
        // is the only host that will read the file. The srvInterface
        // instance passed to this method has a getter for the current
        // node.
        executionNodes.add(srvInterface.getCurrentNodeName());
    }
}
```
// Set the target node(s) in the plan context
planCtxt.setTargetNodes(executionNodes);

// Set parameters for each node reading data that tells it which
// files it will read. In this simple example, just tell it to
// read all of the files the user passed in the file parameter
String files = args.getString("file");

// Get object to write parameters into the plan context object.
ParamWriter nodeParams = planCtxt.getWriter();

// Loop through list of execution nodes, and add a parameter to plan
// context named for each node performing the work, which tells it the
// list of files it will process. Each node will look for a
// parameter named something like "filesFor_vmart_node0002" in its
// prepareUDSources() method.
for (int i = 0; i < executionNodes.size(); i++) {
    nodeParams.setString("filesFor" + executionNodes.get(i), files);
}

// Called on each host that is reading data from a source. This method
// returns an array of UDSource objects that process each source.
@override
public ArrayList<UDSource> prepareUDSources(ServerInterface srvInterface,
                                                 NodeSpecifyingPlanContext planCtxt) throws UdfException {

    // An array to hold the UDSource subclasses that we instantiate
    ArrayList<UDSource> retVal = new ArrayList<UDSource>();

    // Get the list of files this node is supposed to process. This was
    // saved by the plan() method in the planContext.
    String myName = srvInterface.getCurrentNodeName();
    ParamReader params = planCtxt.getReader();
    String filenames = params.getString("filesFor" + myName);

    // Note that you can also be lazy and directly grab the parameters
    // the user passed to the UDSource function in the COPY statement directly
    // by getting parameters from the ServerInterface object. I.e.:

    //String filenames = srvInterface.getParamReader().getString("file");

    // Split comma-separated list into a single list.
    String[] fileList = filenames.split(",");
    for (int i = 0; i < fileList.length; i++) {
        // Instantiate a FileSource object (which is a subclass of UDSource)
        // to read each file. The constructor for FileSource takes the
        // file name of the
        retVal.add(new FileSource(fileList[i]));
    }

    // Return the collection of FileSource objects. They will be called,
    // in turn, to read each of the files.
    return retVal;
}

// Declares which parameters that this factory accepts.
@override
public void getParameterType(ServerInterface srvInterface,
Filter Example: ReplaceCharFilter

The example in this section demonstrates creating a UDFilter that replaces any occurrences of a character in the input stream with another character in the output stream. This example is highly simplified and assumes the input stream is ASCII data.

Always remember that the input and output streams in a UDFilter are actually binary data. If you are performing character transformations using a UDFilter, convert the data stream from a string of bytes into a properly encoded string. For example, your input stream might consist of UTF-8 encoded text. If so, be sure to transform the raw binary being read from the buffer into a UTF string before manipulating it.

Loading and Using the Example

The example UDFilter has two required parameters. The from_char parameter specifies the character to be replaced, and the to_char parameter specifies the replacement character.

Load and use the ReplaceCharFilter UDFilter as follows:

```sql
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaUDLib.jar' ->LANGUAGE 'JAVA';
CREATE LIBRARY => CREATE FILTER ReplaceCharFilter as LANGUAGE 'JAVA' ->name 'com.mycompany.UDL.ReplaceCharFilterFactory' library JavaLib;
CREATE FILTER FUNCTION => CREATE TABLE t (text VARCHAR);
CREATE TABLE => COPY t FROM STDIN WITH FILTER ReplaceCharFilter(from_char='a', to_char='z');
Enter data to be copied followed by a newline. End with a backslash and a period on a line by itself.
=> Mary had a little lamb
=> a man, a plan, a canal, Panama
=> .
=> SELECT * FROM t;
text
-----------------------------
Mzry hzd z little lbmb
z mzn, z plzn, z cznzl, Pznzmbn (2 rows)
```
ReplaceCharFilter], error code: 0
com.vertica.sdk.UdfException: You must supply two parameters to ReplaceChar: 'from_char' and 'to_char'
    at com.vertica.JavaLibs.ReplaceCharFilterFactory.plan(ReplaceCharFilterFactory.java:22)
    at com.vertica.udxfence.UDxExecContext.planUDFilter(UDxExecContext.java:889)
    at com.vertica.udxfence.UDxExecContext.planCurrentUDLType(UDxExecContext.java:865)
    at com.vertica.udxfence.UDxExecContext.planUDL(UDxExecContext.java:821)
    at com.vertica.udxfence.UDxExecContext.run(UDxExecContext.java:242)
    at java.lang.Thread.run(Thread.java:662)

Parser Implementation

The ReplaceCharFilter class reads the data stream, replacing each occurrence of a user-specified character with another character.

```java
package com.vertica.JavaLibs;

import com.vertica.sdk.DataBuffer;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.State.InputState;
import com.vertica.sdk.State.StreamState;
import com.vertica.sdk.UDFilter;

public class ReplaceCharFilter extends UDFilter {
    private byte[] fromChar;
    private byte[] toChar;

    public ReplaceCharFilter(String fromChar, String toChar) {
        // Stores the from char and to char as byte arrays. This is
        // not a robust method of doing this, but works for this simple
        // example.
        this.fromChar = fromChar.getBytes();
        this.toChar = toChar.getBytes();
    }

    @Override
    public StreamState process(ServerInterface srvInterface, DataBuffer input,
                                InputState inputState, DataBuffer output) {

        // Check if there is no more input and the input buffer has been completely
        // processed. If so, filtering is done.
        if (inputState == InputState.END_OF_FILE && input.buf.length == 0) {
            return StreamState.DONE;
        }

        // Get current position in the input buffer
        int offset = output.offset;

        // Determine how many bytes to process. This is either until input
        // buffer is exhausted or output buffer is filled
        int limit = Math.min((input.buf.length - input.offset),
                             (output.buf.length - output.offset));

        for (int i = input.offset; i < limit; i++) {
            // This example just replaces each instance of from_char
            // with to_char. It does not consider things such as multi-byte
        }
    }
}
```
// UTF-8 characters.
if (input.buf[i] == fromChar[0]) {
    output.buf[i+offset] = toChar[0];
} else {
    // Did not find from_char, so copy input to the output
    output.buf[i+offset]=input.buf[i];
}

input.offset += limit;
output.offset += input.offset;

if (input.buf.length - input.offset < output.buf.length - output.offset) {
    return StreamState.INPUT_NEEDED;
} else {
    return StreamState.OUTPUT_NEEDED;
}

Factory Implementation

ReplaceCharFilterFactory requires two parameters (from_char and to_char). The plan() method verifies that these parameters exist and are single-character strings. The method then stores them in the plan context. The prepare() method gets the parameter values and passes them to the ReplaceCharFilter objects, which it instantiates, to perform the filtering.

```java
package com.vertica.JavaLibs;

import java.util.ArrayList;
import java.util.Vector;
import com.vertica.sdk.FilterFactory;
import com.vertica.sdk.PlanContext;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.UDFilter;
import com.vertica.sdk.UdfException;

public class ReplaceCharFilterFactory extends FilterFactory {

    // Run on the initiator node to perform varification and basic setup.
    @Override
    public void plan(ServerInterface srvInterface, PlanContext planCtxt)
        throws UdfException {
        ArrayList<String> args =
            srvInterface.getParamReader().getParamNames();

        // Ensure user supplied two arguments
        if (!args.contains("from_char") && args.contains("to_char")) {
            throw new UdfException(0, "You must supply two parameters +
                " to ReplaceChar: 'from_char' and 'to_char'");
        }
```
// Verify that the from_char is a single character.
String fromChar = srvInterface.getParamReader().getString("from_char");
if (fromChar.length() != 1) {
    String message = String.format("Replacechar expects a single " +
            "character in the 'from_char' parameter. Got length %d",
            fromChar.length());
    throw new UdfException(0, message);
}

// Save the from character in the plan context, to be read by
// prepare() method.
planCtxt.getWriter().setString("fromChar", fromChar);

// Ensure to character parameter is a single character
String toChar = srvInterface.getParamReader().getString("to_char");
if (toChar.length() != 1) {
    String message = String.format("Replacechar expects a single " +
            "character in the 'to_char' parameter. Got length %d",
            toChar.length());
    throw new UdfException(0, message);
}
// Save the to character in the plan data
planCtxt.getWriter().setString("toChar", toChar);

// Called on every host that will filter data. Must instantiate the
// UDFilter subclass.
@Override
public UDFilter prepare(ServerInterface srvInterface, PlanContext planCtxt) throws UdfException {
    // Get data stored in the context by the plan() method.
    String fromChar = planCtxt.getWriter().getString("fromChar");
    String toChar = planCtxt.getWriter().getString("toChar");

    // Instantiate a filter object to perform filtering.
    return new ReplaceCharFilter(fromChar, toChar);
}

// Describe the parameters accepted by this filter.
@Override
public void getParameterType(ServerInterface srvInterface, SizedColumnTypes parameterTypes) {
    parameterTypes.addVarchar(1, "from_char");
    parameterTypes.addVarchar(1, "to_char");
}

Parser Example: Numeric Text

This NumericTextParser example parses integer values spelled out in words rather than
digits (for example "one two three" for one-hundred twenty three). The parser:
• Accepts a single parameter to set the character that separates columns in a row of data. The separator defaults to the pipe (|) character.

• Ignores extra spaces and the capitalization of the words used to spell out the digits.

• Recognizes the digits using the following words: zero, one, two, three, four, five, six, seven, eight, nine.

• Assumes that the words spelling out an integer are separated by at least one space.

• Rejects any row of data that cannot be completely parsed into integers.

• Generates an error, if the output table has a non-integer column.

Loading and Using the Example

Load and use the parser as follows:

```sql
=> CREATE LIBRARY JavaLib AS '/home/dbadmin/JavaLib.jar' LANGUAGE 'JAVA';
CREATE LIBRARY

=> CREATE PARSER NumericTextParser AS LANGUAGE 'java'
->  NAME 'com.myCompany.UDParser.NumericTextParserFactory'
->  LIBRARY JavaLib;
CREATE PARSER FUNCTION
=> CREATE TABLE t (i INTEGER);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
> One
> Two
> One Two Three
> \.
=> SELECT * FROM t ORDER BY i;
  i
-----
 1
 2
123
(3 rows)
=> DROP TABLE t;
DROP TABLE
=> -- Parse multi-column input
=> CREATE TABLE t (i INTEGER, j INTEGER);
CREATE TABLE
=> COPY t FROM stdin WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
> One | Two
> Two | Three
> One Two Three | Four Five Six
```
>> \.
=> SELECT * FROM t ORDER BY i;
  i | j
-----
   1 |  2
   2 |  3
123 | 456
(3 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> -- Use alternate separator character
=> COPY t FROM STDIN WITH PARSER NumericTextParser(separator='*');
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> Five * Six
>> seven * eight
>> nine * one zero
>> \.
=> SELECT * FROM t ORDER BY i;
  i | j
-----
   5 |  6
   7 |  8
   9 | 10
(3 rows)

=> TRUNCATE TABLE t;
TRUNCATE TABLE
=> -- Rows containing data that does not parse into digits is rejected.
=> DROP TABLE t;
DROP TABLE
=> CREATE TABLE t (i INTEGER);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> One Zero Zero
>> Two Zero Zero
>> Three Zed Zed
>> Four Zero Zero
>> Five Zed Zed
>> \.
SELECT * FROM t ORDER BY i;
  i
-----
 100
 200
 400
(3 rows)

=> -- Generate an error by trying to copy into a table with a non-integer column
=> DROP TABLE t;
DROP TABLE
=> CREATE TABLE t (i INTEGER, j VARCHAR);
CREATE TABLE
=> COPY t FROM STDIN WITH PARSER NumericTextParser();
vsq1:UDParse.sql:94: ERROR 3399: Failure in U Dx RPC call
InvokeGetReturnTypeParser(): Error in User Defined Object [NumericTextParser],
Parser Implementation

The following code implements the parser.

```java
package com.myCompany.UDParser;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
import com.vertica.sdk.DataBuffer;
import com.vertica.sdk.DestroyInvocation;
import com.vertica.sdk.RejectedRecord;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.State.InputState;
import com.vertica.sdk.State.StreamState;
import com.vertica.sdk.StreamWriter;
import com.vertica.sdk.UDParser;
import com.vertica.sdk.UdfException;

public class NumericTextParser extends UDParser {

    private String separator; // Holds column separator character

    // List of strings that we accept as digits.
    private List<String> numbers = Arrays.asList("zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine");

    // Hold information about the last rejected row.
    private String rejectedReason;
    private String rejectedRow;

    // Constructor gets the separator character from the Factory's prepare() // method.
    public NumericTextParser(String sepparam) {
        super();
        this.separator = sepparam;
    }

    // Called to perform the actual work of parsing. Gets a buffer of bytes // to turn into tuples.
    @Override
    public StreamState process(ServerInterface srvInterface, DataBuffer input, InputState input_state) throws UdfException, DestroyInvocation {
```
int i = input.offset; // Current position in the input buffer
// Flag to indicate whether we just found the end of a row.
boolean lastCharNewline = false;
// Buffer to hold the row of data being read.
StringBuffer line = new StringBuffer();

// Continue reading until end of buffer.
for (; i < input.buf.length; i++) {
  // Loop through input until we find a linebreak: marks end of row
  char inchar = (char) input.buf[i];
  // Note that this isn’t a robust way to find rows. It should
  // accept a user-defined row separator. Also, the following
  // assumes ASCII line break methods, which isn’t a good idea
  // in the UTF world. But it is good enough for this simple example.
  if (inchar != '\n' && inchar != '\r') {
    // Keep adding to a line buffer until a full row of data is read
    line.append(inchar);
    lastCharNewline = false; // Last character not a new line
  } else {
    // Found a line break. Process the row.
    lastCharNewline = true; // indicate we got a complete row
    // Update the position in the input buffer. This is updated
    // whether the row is successfully processed or not.
    input.offset = i+1;
    // Call processRow to extract values and write tuples to the
    // output. Returns false if there was an error.
    if (!processRow(line)) {
      // Processing row failed. Save bad row to rejectedRow field
      // and return to caller indicating a rejected row.
      rejectedRow = line.toString();
      // Update position where we processed the data.
      return StreamState.REJECT;
    }
    line.delete(0, line.length()); // clear row buffer
  }
}

// At this point, process() has finished processing the input buffer.
// There are two possibilities: need to get more data
// from the input stream to finish processing, or there is
// no more data to process. If at the end of the input stream and
// the row was not terminated by a linefeed, it may need
// to process the last row.

if (input_state == InputState.END_OF_FILE && lastCharNewline) {
  // End of input and it ended on a newline. Nothing more to do
  return StreamState.DONE;
} else if (input_state == InputState.END_OF_FILE && !lastCharNewline) {
  // At end of input stream but didn’t get a final newline. Need to
  // process the final row that was read in, then exit for good.
  if (line.length() == 0) {
    // Nothing to process. Done parsing.
    return StreamState.DONE;
  }
  // Need to parse the last row, not terminated by a linefeed. This
  // can occur if the file being read didn’t have a final line break.
  if (processRow(line)) {
    return StreamState.DONE;
  } else {
private void processRow(StringBuffer line)
    throws UdfException, DestroyInvocation {
String[] columns = line.toString().split(Pattern.quote(separator));
    // Loop through the columns, decoding their contents
for (int col = 0; col < columns.length; col++) {
    // Call decodeColumn to extract value from this column
    Integer colval = decodeColumn(columns[col]);
    if (colval == null) {
        // Could not parse one of the columns. Indicate row should
        // be rejected.
        return false;
    }
    // Column parsed OK. Write it to the output. writer is a field
    // provided by the parent class. Since this parser only accepts
    // integers, there is no need to verify that data type of the parsed
    // data matches the data type of the column being written. In your
    // UDFParsers, you may want to perform this verification.
    writer.setLong(col, colval);
}
// Done with the row of data. Advance output to next row.

// Note that this example does not verify that all of the output columns
// have values assigned to them. If there are missing values at the
// end of a row, they get automatically get assigned a default value
// (0 for integers). This isn't a robust solution. Your UDFParser
// should perform checks here to handle this situation and set values
// (such as null) when appropriate.
writer.next();
return true; // Successfully processed the row.
}

// Gets a string with text numerals, i.e. "One Two Five Seven" and turns
// it into an integer value, i.e. 1257. Returns null if the string could not
// be parsed completely into numbers.
private Integer decodeColumn(String text) {
    int value = 0; // Hold the value being parsed.
    // Split string into individual words. Eat extra spaces.
    String[] words = text.toLowerCase().trim().split("\\s+");
// Loop through the words, matching them against the list of
digit strings.
for (int i = 0; i < words.length; i++) {
    if (numbers.contains(words[i])) {
        // Matched a digit. Add the it to the value.
        int digit = numbers.indexOf(words[i]);
        value = (value * 10) + digit;
    } else {
        // The string didn’t match one of the accepted string values
        // for digits. Need to reject the row. Set the rejected
        // reason string here so it can be incorporated into the
        // rejected reason object.
        //
        // Note that this example does not handle null column values.
        // In most cases, you want to differentiate between an
        // unparsable column value and a missing piece of input
        // data. This example just rejects the row if there is a missing
        // column value.
        rejectedReason = String.format(
            "Could not parse '%s' into a digit",words[i]);
        return null;
    }
}
return value;

// Vertica calls this method if the parser rejected a row of data
// to find out what went wrong and add to the proper logs. Just gathers
// information stored in fields and returns it in an object.
@Override
public RejectedRecord getRejectedRecord() throws UdfException {
    return new RejectedRecord(rejectedReason,rejectedRow.toCharArray(),
        rejectedRow.length(), "\n");
}

ParserFactory Implementation

The following code implements the parser factory.

NumericTextParser accepts a single optional parameter named separator. This
parameter is defined in the getParameterType() method, and the plan() method stores
its value. NumericTextParser outputs only integer values. Therefore, if the output table
contains a column whose data type is not integer, the getParserReturnType() method
throws an exception.

package com.myCompany.UDParser;
import java.util.regex.Pattern;
import com.vertica.sdk.ParamReader;
import com.vertica.sdk.ParamWriter;
import com.vertica.sdk.ParserFactory;
import com.vertica.sdk.PerColumnParamReader;
import com.vertica.sdk.PlanContext;
import com.vertica.sdk.ServerInterface;
import com.vertica.sdk.SizedColumnTypes;
import com.vertica.sdk.UDParser;
import com.vertica.sdk.UdfException;
import com.vertica.sdk.VerticaType;

public class NumericTextParserFactory extends ParserFactory {

    // Called once on the initiator host to check the parameters and set up the
    // context data that hosts performing processing will need later.
    @Override
    public void plan(ServerInterface srvInterface,
                     PerColumnParamReader perColumnParamReader,
                     PlanContext planCtxt) {
        String separator = "|"; // assume separator is pipe character

        // See if a parameter was given for column separator
        ParamReader args = srvInterface.getParamReader();
        if (args.containsParameter("separator")) {
            separator = args.getString("separator");
            if (separator.length() > 1) {
                throw new UdfException(0,
                        "Separator parameter must be a single character");
            }
            if (!Pattern.quote(separator).matches("[a-zA-Z]")) {
                throw new UdfException(0,
                        "Separator parameter cannot be a letter");
            }
        }

        // Save separator character in the Plan Data
        ParamWriter context = planCtxt.getWriter();
        context.setString("separator", separator);
    }

    // Define the data types of the output table that the parser will return.
    // Mainly, this just ensures that all of the columns in the table which
    // is the target of the data load are integer.
    @Override
    public void getParserReturnType(ServerInterface srvInterface,
                                     PerColumnParamReader perColumnParamReader,
                                     PlanContext planCtxt,
                                     SizedColumnTypes argTypes,
                                     SizedColumnTypes returnType) {

        // Get access to the output table's columns
        for (int i = 0; i < argTypes.getColumnCount(); i++) {
            if (argTypes.getColumnType(i).isInt()) {
                // Column is integer... add it to the output
                returnType.addInt(argTypes.getColumnName(i));
            } else {
                // Column isn't an int, so throw an exception.
                // Technically, not necessary since the
                // UDx framework will automatically error out when it sees a
                // Discrepancy between the type in the target table and the
                // types declared by this method. Throwing this exception will
                // provide a clearer error message to the user.
            }
        }
    }
}
String message = String.format(
   "Column %d of output table is not an Int", i + 1);
throw new UdfException(0, message);
} }
}

// Instantiate the UDParse subclass named NumericTextParser. Passes the
// separator character as a parameter to the constructor.
@Override
public UDParse prepare(ServerInterface srvInterface,
   PerColumnParamReader perColumnParamReader, PlanContext planCtxt,
   SizedColumnTypes returnType) throws UdfException {
   // Get the separator character from the context
   String separator = planCtxt.getReader().getString("separator");
   return new NumericTextParser(separator);
}

// Describe the parameters accepted by this parser.
@Override
public void getParameterType(ServerInterface srvInterface,
   SizedColumnTypes parameterTypes) {
   parameterTypes.addVarchar(1, "separator");
}
}

Parser Example: JSON Parser

The JSON Parser consumes a stream of JSON objects. Each object must be well formed and on
a single line in the input. Use line breaks to delimit the objects. The parser uses the field names
as keys in a map, which become column names in the table. You can find the code for this
example in /opt/vertica/packages/flextable/examples. This directory also contains an example
data file.

This example uses the setRowFromMap() method to write data.

Loading and Using the Example

1. Load the library and define the JSON parser, using the third-party library (gson-2.2.4.jar).
   See the comments in JsonParser.java for a download URL.
2. Define a table, and then use the JSON parser to load data into that table.

```sql
--> CREATE TABLE mountains(name varchar(64), type varchar(32), height integer);
CREATE TABLE

--> COPY mountains FROM '/opt/vertica/packages/flextable/examples/mountains.json'
--> WITH PARSER JsonParser();
-[- RECORD 1 ]----
Rows Loaded | 2

--> SELECT * from mountains;
-[- RECORD 1 ]-------
name | Everest
    | mountain
height | 29029
-[- RECORD 2 ]-------
name | Mt St Helens
    | volcano
height | 
```

The data file contains a value (hike_safety) that was not loaded because the table definition did not include that column. The data file follows:

```json
{ "name": "Everest", "type":"mountain", "height": 29029, "hike_safety": 34.1 }
{ "name": "Mt St Helens", "type": "volcano", "hike_safety": 15.4 }
```

Implementation

The following code shows the `process()` method from `JsonParser.java`. The parser attempts to read the input into a Map. If the read is successful, the JSON Parser calls `setRowFromMap()`:

```java
@Override
public StreamState process(ServerInterface srvInterface, DataBuffer input,
    InputState inputState) throws UdfException, DestroyInvocation {
    clearReject();
    StreamWriter output = getStreamWriter();
    
    while (input.offset < input.buf.length) {
        ByteBuffer lineBytes = consumeNextLine(input, inputState);
        if (lineBytes == null) {
            return StreamState.INPUT_NEEDED;
        }
        
        String lineString = StringUtils.newString(lineBytes);
        try {
            Map<String,Object> map = gson.fromJson(lineString, parseType);
            if (map == null) {
```
continue;
}

output.setRowFromMap(map);
// No overrides needed, so just call next() here.
output.next();
} catch (Exception ex) {
    setReject(lineString, ex);
    return StreamState.REJECT;
}

The factory, JsonParserFactory.java, instantiates and returns a parser in the prepare() method. No additional setup is required.
Connecting to Vertica

This book explains several methods of connecting to Vertica, including:

- Directly connecting to Vertica using the vsql client application.
- Installing and configuring the Vertica client libraries to allow client applications to access Vertica.
- Developing your own client applications using the Vertica client libraries.
Using vsql

vsql is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.

If you are using the vsql client installed on the server, then you can connect from the:

- Administration Tools
- Linux command line

You can also install the vsql client for other supported platforms.

A man page is available for vsql. If you are running as the dbadmin user, simply type: `man vsql`. If you are running as a different user, type: `man -M /opt/vertica/man vsql`.

General Notes

- SQL statements can be spread over several lines for clarity.

- vsql can handle input and output in UTF-8 encoding. The terminal emulator running vsql must be set up to display the UTF-8 characters correctly. Follow the documentation of your terminal emulator. The following example shows the settings in PuTTy from the Change Settings > Window > Translation option:
See also Best Practices for Working with Locales.

- Cancel SQL statements by typing Ctrl+C.
- Traverse command history by typing Ctrl+R.
- When you disconnect a user session, any transactions in progress are automatically rolled back.
- To view wide result sets, use the Linux less utility to truncate long lines.
  a. Before connecting to the database, specify that you want to use less for query output:

```sh
$ export PAGER=less
```
  b. Connect to the database.
  c. Query a wide table:

```sql
=> select * from wide_table;
```
d. At the less prompt, type:

```
-s
```

If a shell running vsq1 fails (crashes or freezes), the vsq1 processes continue to run even if you stop the database. In that case, log in as root on the machine on which the shell was running and manually kill the vsq1 process. For example:

```
# ps -ef | grep vertica
  fred 2401 1 0 06:02 pts/1 00:00:00 /opt/vertica/bin/vsql -p 5433 -h test01_site01 quick_start_single

# kill -9 2401
```
## Installing the vsq1 Client

The vsq1 client is installed as part of the Vertica server rpm. It is also available as a download for other Unix-based systems such as HP-UX, AIX, and Mac OSX.

### Unix Installation

1. Use a web browser to log in to the myVertica portal.
2. Click Downloads, and choose Client Drivers.
3. Download the appropriate vsq1 client. There are both 32-bit and 64-bit versions for most platforms.

   **Important:** Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsq1 and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, Installing the FIPS Client Driver for ODBC and vsq1.

4. Extract the tarball. The tarball is organized to extract into /opt/vertica if you extract it at the root (/) of the drive.
5. Optionally add the directory where the vsq1 client resides to your path.
6. Verify mode on the vsq1 file is executable. For example:  `chmod ugo+x vsq1_VERSION`  
7. Set your shell locale to a locale supported by vsq1. For example, in your .profile, add,  
   ```
   export LANG=en_US.UTF-8
   ```

### Windows Installation

vsq1 on Windows is installed as part of the Windows Client Driver package. For installation details, see [The Vertica Client Drivers and Tools for Windows](#).

See [Using vsq1 for Windows Users](#) for details on using vsq1 in a Windows console.
Connecting From the Administration Tools

You can use the Administration Tools to connect to a database using vsq1 on any node in the cluster.

1. Log in as the database administrator user; for example, dbadmin.

   Note: Vertica does not allow users with root privileges to connect to a database for security reasons.

2. Run the Administration Tools.

   /opt/vertica/bin/admintools

3. On the Main Menu, select Connect to Database.

   Main Menu
   
   1. View Database Cluster State
   2. Connect to Database
   3. Start Database
   4. Stop Database
   5. Restart Vertica on Host
   6. Configuration Menu
   7. Advanced Menu
   8. Help Using the Administration Tools
   E. Exit

4. Supply the database password if asked:

   Password:

   When you create a new user with the CREATE USER command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the ALTER USER command.

5. The Administration Tools connect to the database and transfer control to vsq1.
Welcome to vsql, the Vertica Analytic Database interactive terminal.
Type: \h or \? for help with vsql commands
\g or terminate with semicolon to execute query
\q to quit

Note: See Meta-Commands for the various commands you can run while connected to the database through the Administration Tools.
Connecting from the Command Line

You can connect to a database using vsq1 from the command line on multiple client platforms. If the connection cannot be made for any reason—for example, you have insufficient privileges, or the server is not running on the targeted host—vsq1 returns an error and terminates.

Syntax

```
/opt/vertica/bin/vsql [-h host] [ option...] [ dbname [ username ]]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>host</code></td>
<td>Optional if you connect to a local server. You can provide an IPv4 or IPv6 IP address or a host name. For Vertica servers that have both IPv4 and IPv6 addressed and you have provided a host name instead of an IP address, you can prefer to use an IPv4 address with the <code>-4</code> option and to use the IPv6 address with the <code>-6</code> option if the DNS is configured to provide both IPv4 and IPv6 addresses. If you are using IPv6 and provide an IP address, you must append the address with an <code>%interface name</code>.</td>
</tr>
<tr>
<td><code>option</code></td>
<td>One or more vsq1 Command-Line Options If the database is password protected, you must specify the <code>-w</code> or <code>--password</code> command line option.</td>
</tr>
<tr>
<td><code>dbname</code></td>
<td>The name of the target database, by default your Linux user name.</td>
</tr>
<tr>
<td><code>username</code></td>
<td>A database username, by default your Linux user name.</td>
</tr>
</tbody>
</table>

Exit Codes

vsq1 returns 0 to the shell when it terminates normally. Otherwise, it returns one of the following:
1: A fatal error occurred—for example, out of memory or file not found.

2: The connection to the server went bad and the session was not interactive

3: An error occurred in a script and the variable ON_ERROR_STOP was set.

Unrecognized words in the command line might be interpreted as database or user names.

Examples

The following example shows how to capture error messages by redirecting vsq1 output to the output file retail_queries.out:

$ vsq1 --echo-all < retail_queries.sql > retail_queries.out 2>&1

Command-Line Options

This section contains the command-line options for vsq1.

General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--command command</td>
<td>Runs one command and exits. This command is useful in shell scripts.</td>
</tr>
<tr>
<td>-c command</td>
<td></td>
</tr>
<tr>
<td>--dbname dbname</td>
<td>Specifies the name of the database to which you want to connect. Using this</td>
</tr>
<tr>
<td>-d dbname</td>
<td>command is equivalent to specifying dbname as the first non-option argument</td>
</tr>
<tr>
<td></td>
<td>on the command line.</td>
</tr>
<tr>
<td>--file filename</td>
<td>Uses the filename as the source of commands instead of reading commands</td>
</tr>
<tr>
<td>-f filename</td>
<td>interactively. After the file is processed, vsq1 terminates.</td>
</tr>
<tr>
<td>--help</td>
<td>Displays help about vsq1 command line arguments and exits.</td>
</tr>
<tr>
<td>--timing -i</td>
<td>Enables the \timing meta-command.</td>
</tr>
</tbody>
</table>
### --list
-1

Returns all available databases, then exits. Other non-connection options are ignored. This command is similar to the internal command \list.

### --set assignment
--variable assignment
-v assignment

Performs a variable assignment, like the \set internal command.

### --version
-V

Prints the vsql version and exits.

### --no-vsqlr
-X

Disables all command line editing and history functionality.

## Connection Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>When resolving hostnames in dual stack environments, prefer IPv4 addresses.</td>
</tr>
<tr>
<td>-6</td>
<td>When resolving hostnames in dual stack environments, prefer IPv6 addresses.</td>
</tr>
<tr>
<td>-B server:port[,]...</td>
<td>Sets connection backup server/port. Use comma-separated multiple hosts (default: not set). If using an IPv6 address, enclose the address in brackets ([, ]) and place the port outside of the brackets. For example \B [260:0:a13:8a4:9d9f:e0e3:1181:7f51]:5433</td>
</tr>
<tr>
<td>--enable-connection</td>
<td>Enables connection load balancing (default: not enabled).</td>
</tr>
<tr>
<td>-C</td>
<td></td>
</tr>
<tr>
<td>--host hostname</td>
<td>Specifies the host name of the machine on which the server is running.</td>
</tr>
<tr>
<td>-h hostname</td>
<td></td>
</tr>
<tr>
<td>-k krb-service</td>
<td>Provides the service name portion of the Kerberos principal</td>
</tr>
</tbody>
</table>

**Note:** You can only use load balancing with one address family in dual stack environments. For example, if you've configured load balancing for IPv6 addresses, then when an IPv4 client connects and requests load balancing the server does not allow it.
Using -k is equivalent to using the drivers' KerberosServiceName connection string.

- **K krb-host**
  Provides the instance or host name portion of the Kerberos principal. -K is equivalent to the drivers' KerberosHostName connection string.

- **--sslmode --m**
  Specifies the policy for making SSL connections to the server. Options are require, prefer, allow, and disable. You can also set the VSQL_SSLMODE variable to achieve the same effect. If the variable is set, the command-line option overrides it.

- **--port port --p port**
  Specifies the TCP port or the local socket file extension on which the server is listening for connections. Defaults to port 5433.

- **--username username --U username**
  Connects to the database as the user username instead of the default.

- **-w password**
  Specifies the password for a database user.

  **Note:** Using this command-line option displays the database password in plain text. Use it with care, particularly if you are connecting as the database administrator, to avoid exposing sensitive information.

- **--password --w**
  Forces vsql to prompt for a password before connecting to a database. The password is not displayed on the screen. This option remains set for the entire session, even if you change the database connection with the meta-command \connect.

### Output Formatting

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--no-align -A</td>
<td>Switches to unaligned output mode. (The default output mode is aligned.)</td>
</tr>
<tr>
<td>-b</td>
<td>Beep on command completion.</td>
</tr>
<tr>
<td>--field-separator</td>
<td>Specifies the field separator for unaligned output (default: Vertica)</td>
</tr>
</tbody>
</table>
| separator  | "|"
| -F separator | (See -A --no-align.) Using this command is equivalent to \pset fieldsep or \f.

| --html|--H | Turns on HTML tabular output. Using this command is equivalent to using the \pset format html or the \H command.

| --pset assignment --P assignment | Lets you specify printing options in the style of \pset on the command line. You must separate the name and value with an equals (=) sign instead of a space. Thus, to set the output format to LaTeX, you could write -P format=latex.

| -Q | Turns on trailing record separator. Use \pset trailingrecordsep to toggle the trailing record separator on or off.

| --record-separator separator --R separator | Uses separator as the record separator. Using this command is equivalent to using the \pset recordsep command.

| --tuples-only --t | Disables printing of column names, result row count footers, and so on. This is equivalent to the \t command.

| --table-attr options --T options | Allows you to specify options to be placed within the HTML table tag. See \pset for details.

| --expanded --x | Enables extended table formatting mode. This is equivalent to the command \x.

### Input and Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--echo-all -a</td>
<td>Prints all input lines to standard output as they are read. This approach is more useful for script processing than interactive mode. It is the same as setting the variable ECHO to all.</td>
</tr>
<tr>
<td>--echo-queries -e</td>
<td>Copies all SQL commands sent to the server to standard output. Using this command is equivalent to setting the variable ECHO to queries.</td>
</tr>
</tbody>
</table>
-E | Displays queries generated by internal commands.
-n | Disables command line editing.
--output filename | Writes all query output into file filename. Using this command is equivalent to using the command `\o`.
-o filename | Using this command is equivalent to using the command `\o`.
--quiet | Specifies that vsql do its work quietly (without informational output, such as welcome messages). This command is useful with the -c option. Within vsql you can also set the QUIET variable to achieve the same effect.
--single-step | Runs in single-step mode for debugging scripts. Forces vsql to prompt before each statement is sent to the database and allows you to cancel execution.
-m | Runs in single-line mode where a newline terminates a SQL command, as if you are using a semicolon.

Note: This mode is provided only by customer request. Vertica recommends that you not use single-line mode in cases where you mix SQL and meta-commands on a line. In single-line mode, the order of execution might be unclear to the inexperienced user.

-A --no-align
-A or --no-align switches to unaligned output mode. The default output mode is aligned.

-a --echo-all
-a or --echo-all prints all input lines to standard output as they are read. This is more useful for script processing than interactive mode. It is equivalent to setting the variable `ECHO` to all.

-c --command
-c command or --command command runs one command and exits. This is useful in shell scripts.
Use either:
A command string that can be completely parsed by the server that does not contain features specific to vsqld

A single meta-command

You cannot mix SQL and vsqld meta-commands. You can, however, pipe the string into vsqld as shown:

```bash
echo "\\timing\\select * from t" | ../Linux64/bin/vsqld
Timing is on.
i | c | v
---- ---- ----
(0 rows)
```

Note: If you use double quotes (") with echo, you must double the backslashes (\).

-d --dbname

-d db-name or --dbname db-name specifies the name of the database to connect to. This is equivalent to specifying db-name as the first non-option argument on the command line.

-E

-E displays queries generated by internal commands.

-e --echo-queries

-e --echo-queries copies all SQL commands sent to the server to standard output as well. This is equivalent to setting the variable ECHO to queries.

-F --field-separator

-F separator or --field-separator separator specifies the field separator for unaligned output (default: "\") (-P fieldsep=). (See -A --no-align.) This is equivalent to \psset fieldsep or \f.

To set the field separator value to a control character, use your shell's control character escape notation. In Bash, you specify a control character in an argument using a dollar sign ($) followed by a string contained in single quotes. This string can contain C-string escapes...
(such as \t for tab), or a backslash (\) followed by an octal value for the character you want to use.

The following example demonstrates setting the separator character to tab (\t), vertical tab (\v) and the octal value of vertical tab (\013).

```
$ vsql -At -c "SELECT * FROM testtable;"
A|1|2|3  
B|4|5|6

$ vsql -F $'\t' -At -c "SELECT * FROM testtable;"
A 1 2 3  
B 4 5 6

$ vsql -F $'\v' -At -c "SELECT * FROM testtable;"
A 1  
  2  
  3  
B 4  
  5  
  6

$ vsql -F $'\013' -At -c "SELECT * FROM testtable;"
A 1  
  2  
  3  
B 4  
  5  
  6
```

**-f --file**

\( -f \) **filename** or **--file filename** uses `filename` as the source of commands instead of reading commands interactively. After the file is processed, vsql terminates.

If `filename` is a hyphen (-), standard input is read.

Using this option is different from writing vsql < `filename`. Using -f enables some additional features such as error messages with line numbers. Conversely, the variant using the shell's input redirection should always yield exactly the same output that you would have gotten had you entered everything manually.

**? --help**

\( -? \) **--help** displays help about vsql command line arguments and exits.
-H --html

-H --html turns on HTML tabular output. This is equivalent to \pset format html or the \H command.

-h --host

-h hostname or --host hostname specifies the host name of the machine on which the server is running.

The following requirements and restrictions apply:

- If you use client authentication with a Kerberos connection method of either gss or krb5, you must specify -h hostname.

- If you use client authentication with a "local" connection type specified, and you want to match the client authentication entry, do not use -h hostname.

-i -- timing

Enables the \timing meta-command. You can only use this command with the -c --command and -f --file commands:

```
$VSQL -h host1 -U user1 -d VMart -p 15 -w ***** -i -f transactions.sql
```

For more information, see \timing.

-l --list

-l or --list returns all available databases, then exits. Other non-connection options are ignored. This command is similar to the internal command \list.

-m --sslmode

-m or --sslmode specifies the policy for making SSL connections to the server. Options are verify_full, verify_ca require, prefer, allow, and disable. You can also set the VSQL_SSLMODE variable to achieve the same effect. If the variable is set, the command-line option overrides it.
For information on these modes see Configuring SSL for ODBC Clients.

-n
-n disables command line editing.

-o --output
-o filename or --output filename writes all query output into file filename. This is equivalent to the command \o.

-P --pset
-P assignment or --pset assignment lets you specify printing options in the style of \pset on the command line. Note that you have to separate name and value with an equal sign instead of a space. Thus to set the output format to LaTeX, you could write -P format=latex.

-p --port
-p port or --port port specifies the TCP port or the local socket file extension on which the server is listening for connections. Defaults to port 5433.

-q --quiet
-q or --quiet specifies that vsql do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this appears. This is useful with the -c option. Within vsql you can also set the QUIET variable to achieve the same effect.

-R --record-separator
-R separator or --record-separator separator specifies separator as the record separator. This is equivalent to the \pset recordsep command.
-S --single-line

-S --single-line runs in single-line mode where a newline terminates a SQL command, like the semicolon does.

Note: This mode is provided for those who insist on it, but you are not necessarily encouraged to use it, particularly if you mix SQL and meta-commands on a line. The order of execution might not always be clear to the inexperienced user.

-s --single-step

-s --single-step runs in single-step mode for debugging scripts. Forces vsql to prompt before each statement is sent to the database and allows you to cancel execution.

-T --table-attr

-T table-options or --table-attr table-options lets you specify options to be placed within the HTML table tag. See \pset for details.

-t --tuples-only

-t or --tuples-only disables printing of column names, result row count footers, and so on. This is equivalent to the \t command.

-V --version

-V or --version prints the vsql version and exits.

-v --variable --set

-v assignment, --variable assignment, and --set assignment perform a variable assignment, like the internal command \set.

Note: You must separate name and value, if any, by an equals sign (=) on the command line.
To unset a variable, omit the equal sign. To set a variable without a value, use the equals sign but omit the value. Make these assignments at a very early stage of start-up, so that variables reserved for internal purposes can get overwritten later.

-X --no-vsq1rc

-X --no-vsq1rc prevents the start-up file from being read: the system-wide vsq1rc file or the user’s ~/.vsq1rc file.

-x --expanded

-x or --expanded enables extended table formatting mode. This is equivalent to the command \x.

Connecting From a Non-Cluster Host

You can use the Vertica vsq1 executable image on a non-cluster Linux host to connect to a Vertica database.

- On Red Hat, CentOS, and SUSE systems, you can install the client driver RPM, which includes the vsq1 executable. See Installing the Client RPM on Red Hat and SUSE for details.

- If the non-cluster host is running the same version of Linux as the cluster, copy the image file to the remote system. For example:

$ scp host01:/opt/vertica/bin/vsql .$ ./vsql

- If the non-cluster host is running a different version of Linux than your cluster hosts, and that operating system is not Red Hat version 5 64-bit or SUSE 10/11 64-bit, you must install the Vertica server RPM in order to get vsq1. Download the appropriate rpm package from the Download tab of the myVertica portal then log into the non-cluster host as root and install the rpm package using the command:

# rpm -Uvh filename

In the above command, filename is the package you downloaded. Note that you do not have to run the install_Vertica script on the non-cluster host in order to use vsq1.
Notes

- Use the same Command-Line Options that you would on a cluster host.

- You cannot run vsq1 on a Cygwin bash shell (Windows). Use ssh to connect to a cluster host, then run vsq1.
Meta-Commands

Anything you enter in vsql that begins with an unquoted backslash is a vsql meta-command that is processed by vsql itself. These commands help make vsql more useful for administration or scripting. Meta-commands are more commonly called slash or backslash commands.

The format of a vsql command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace into an argument you can quote it with a single quote. To include a single quote into such an argument, precede it by a backslash. Anything contained in single quotes is furthermore subject to C-like substitutions for \n (new line), \t (tab), \digits, \digits, and \x digits (the character with the given decimal, octal, or hexadecimal code).

If an unquoted argument begins with a colon (:), it is taken as a vsql variable and the value of the variable is used as the argument instead.

Arguments that are enclosed in backquotes (`) are taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) is taken as the argument value. The above escape sequences also apply in backquotes.

Some commands take a SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (") protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, FOO"BAR"BAZ is interpreted as fooBARbaz, and "A weird" name" becomes A weird name.

Parsing for arguments stops when another unquoted backslash occurs. This is taken as the beginning of a new meta-command. The special sequence `\` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and vsql commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

`! [ COMMAND ]`

`! [ COMMAND ]` executes a command in a Linux shell (passing arguments as entered) or starts an interactive shell.
\? displays help information about the meta-commands. Works the same as \h.

```plaintext
=> \?

General
- \[connect] [DBNAME] - [USER]
  connect to new database (currently "VMart")
- \cd [DIR]
  change the current working directory
- \q
  quit vsql
- \set [NAME] [VALUE]
  set internal variable, or list all if no parameters
- \! [COMMAND]
  execute command in shell or start interactive shell
- \password [USER]
  change user's password

Query Buffer
- \e [FILE]
  edit the query buffer (or file) with external editor
- \g FILE
  send query buffer to server
- \g [COMMAND]
  send query buffer to server and results to file
- \g [COMMAND]
  send query buffer to server and pipe results to command
- \p
  show the contents of the query buffer
- \r
  reset (clear) the query buffer
- \s [FILE]
  display history or save it to file
- \w FILE
  write query buffer to file

Input/Output
- \echo [STRING]
  write string to standard output
- \i FILE
  execute commands from file
- \o FILE
  send all query results to file
- \o [COMMAND]
  pipe all query results to command
- \o
  close query-results file or pipe
- \qecho [STRING]
  write string to query output stream (see \o)

Informational
- \d [PATTERN]
  describe tables (list tables if no argument is supplied)
- \df [PATTERN]
  list functions
- \dj [PATTERN]
  list projections
- \dn [PATTERN]
  list schemas
- \dp [PATTERN]
  list table access privileges
- \ds [PATTERN]
  list sequences
- \ds [PATTERN]
  list system tables
- \dt [PATTERN]
  list tables
- \dvt [PATTERN]
  list tables and views
- \dT [PATTERN]
  list data types
- \du [PATTERN]
  list users
- \dv [PATTERN]
  list views
- \l
  list all databases
- \z [PATTERN]
  list table access privileges (same as \dp)

Formatting
- \a
  toggle between unaligned and aligned output mode
- \b
  toggle beep on command completion
- \C [STRING]
  set table title, or unset if none
- \f [STRING]
  show or set field separator for unaligned query output
- \H
  toggle HTML output mode (currently off)
- \pset NAME [VALUE]
```

set table output option
(NAME := \{format|border|expanded|fieldsep|footer|null|recordsep|tuples_only|title|tableattr|pager\})
\t show only rows (currently off)
\T [STRING] set HTML \{table\} tag attributes, or unset if none
\x toggle expanded output (currently off)

\a
\a toggles output format alignment. This command is kept for backwards compatibility. See \pset for a more general solution.
\a is similar to the command line option -A --no-align, which only disables alignment.

\b
\b toggles beep on command completion.

\c (or \connect) [ Dbname [ Username ] ]
\c (or \connect) [ dbname [ username ] ] establishes a connection to a new database and/or under a user name. The previous connection is closed. If dbname is - the current database name is assumed.
If username is omitted the current user name is assumed.
As a special rule, \connect without any arguments connects to the default database as the default user (as you would have gotten by starting vsql without any arguments).
If the connection attempt fails (wrong user name, access denied, or other error messages), the previous connection is kept if and only if vsql is in interactive mode. When executing a non-interactive script, processing immediately stops with an error. This is a distinction that avoids typos and prevents scripts from accidentally acting on the wrong database.

\C [ STRING ]
\C [ STRING ] sets the title of any tables being printed as the result of a query or unsets any such title. This command is equivalent to \pset title \textit{title}. (The name of this command derives from "caption," as it was previously only used to set the caption in an HTML table.)
\cd [ DIR ]

\cd [ DIR ] changes the current working directory to directory. Without argument, changes to the current user's home directory.

To print your current working directory, use \! pwd. For example:

```plaintext
gp> \! pwd
/home/dbadmin
```

The \d [ PATTERN ] Meta-Commands

This section describes the various \d meta-commands.

All \d meta-commands take an optional pattern (asterisk [ * ] or question mark [ ? ]) and return only the records that match that pattern.

The ? argument is useful if you can't remember if a table name uses an underscore or a dash:

```plaintext
gp> \d v?internal
  List of schemas
  Name    | Owner
-----------+----------
 v_internal | dbadmin
(1 row)
```

The output from the \d metacommands places double quotes around non-alphanumeric table names and table names that are keywords, such as in the following example.

```plaintext
gp> CREATE TABLE my_keywords.precisio(x numeric(4,2));
CREATE TABLE
gp> \d
  List of tables
  Schema | Name       | Kind | Owner
-----------+------------+------+----------
 my_keywords | "precision" | table | dbadmin
```

Double quotes are optional when you use a \d command with pattern matching.

\d [ PATTERN ]

The \d meta-command lists all tables in the database and returns their schema, table name, kind (e.g., table), and owner.
If you use `\d [ PATTERN ]` and provide the schema name or table name (or wildcard or ? characters) as the pattern argument, the result shows more detailed information about the tables:

- Schema name
- Table name
- Column name
- Column data type
- Data type size
- Default column value
- Whether the column accepts null values or has a NOT NULL constraint
- Whether there is a primary key or foreign key constraint

To view information about system tables, you must include the V_MONITOR or V_CATALOG as the pattern argument; for example:

```
\d v_catalog.types  -- information on the types table in v_catalog schema
\d v_catalog.*      -- information on all table columns in v_catalog schema
```

The following output is the result of all tables in the vmart schema, which is in the PUBLIC schema.

```
VMart=> \d
List of tables
+---------+-----------------+----+--------+
<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>online_sales</td>
<td>call_center_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>online_sales</td>
<td>online_page_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>online_sales</td>
<td>online_sales_fact</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>customer_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>date_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>employee_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>inventory_fact</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>product_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>promotion_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>shipping_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>vendor_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>warehouse_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>store</td>
<td>store_dimension</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>store</td>
<td>store_orders_fact</td>
<td>table</td>
<td>dbadmin</td>
</tr>
<tr>
<td>store</td>
<td>store_sales_fact</td>
<td>table</td>
<td>dbadmin</td>
</tr>
</tbody>
</table>
(15 rows)
```

This example returns information on the inventory_fact table in the VMart database:
VMart => \x
Expanded display is on.
VMart => \d inventory_fact
List of Fields by Tables

<table>
<thead>
<tr>
<th>RECORD 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

Use the question mark [ ? ] argument to replace a single character. For example, the ? argument replaces the last character in the user-created SubQ1 and SubQ2 tables, so the command returns information about both:
=> \d SubQ?

### List of Fields by Tables

<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Size</th>
<th>Default</th>
<th>Not Null</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>SubQ1</td>
<td>a</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>SubQ1</td>
<td>b</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>SubQ1</td>
<td>c</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>SubQ2</td>
<td>x</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>SubQ2</td>
<td>y</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td>SubQ2</td>
<td>z</td>
<td>int</td>
<td>8</td>
<td>f</td>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(6 rows)

If you run the \d command and provide both the schema and table name, output includes columns for tables that match the pattern

VMart=> \x
Expanded display is on.
VMart=> \d v_catalog.types
List of Fields by Tables
-[- RECORD 1 ]-------------------
  | Schema | v_catalog
  | Table   | types
  | Column  | column_size
  | Type    | int
  | Size    | 8
  | Default |         |
  | Not Null| f       |
  | Primary Key | f     |
  | Foreign Key |      |
-[- RECORD 2 ]-------------------
  | Schema | v_catalog
  | Table   | types
  | Column  | creation_parameters
  | Type    | varchar(128)
  | Size    | 128
  | Default |         |
  | Not Null| f       |
  | Primary Key | f     |
  | Foreign Key |      |
-[- RECORD 3 ]-------------------
  | Schema | v_catalog
  | Table   | types
  | Column  | epoch
  | Type    | int
  | Size    | 8
  | Default |         |
  | Not Null| f       |
  | Primary Key | f     |
  | Foreign Key |      |
-[- RECORD 4 ]-------------------
  | Schema | v_catalog
  | Table   | types
  | Column  | interval_mask
  | Type    | int
  | Size    | 8
<p>| Default |         |
| Not Null| f       |</p>
<table>
<thead>
<tr>
<th>Primary Key</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreign Key</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
<tr>
<td>Primary Key</td>
</tr>
<tr>
<td>Foreign Key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RECORD 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>Table</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>Not Null</td>
</tr>
</tbody>
</table>
Primary Key | f
Foreign Key |  

To view all tables in a schema, use the wildcard character. The following command would return all system tables in the V_CATALOG schema:

```sql
=> \d v_catalog.*
```

### \df [ PATTERN ]

The `\df` [ PATTERN ] meta-command returns all function names, the function return data type, and the function argument data type. Also returns the procedure names and arguments for all procedures that are available to the user.

<table>
<thead>
<tr>
<th>procedure_name</th>
<th>procedure_return_type</th>
<th>procedure_argument_types</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Float</td>
<td>Float</td>
</tr>
<tr>
<td>abs</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>abs</td>
<td>Interval</td>
<td>Interval</td>
</tr>
<tr>
<td>abs</td>
<td>Interval</td>
<td>Interval</td>
</tr>
<tr>
<td>abs</td>
<td>Numeric</td>
<td>Numeric</td>
</tr>
<tr>
<td>acos</td>
<td>Float</td>
<td>Float</td>
</tr>
</tbody>
</table>

... (2 rows)

The following example uses the wildcard character to search for all functions that begin with `as`:

```sql
vmartdb=> \df as*
```

<table>
<thead>
<tr>
<th>procedure_name</th>
<th>procedure_return_type</th>
<th>procedure_argument_types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascii</td>
<td>Integer</td>
<td>Varchar</td>
</tr>
<tr>
<td>asin</td>
<td>Float</td>
<td>Float</td>
</tr>
</tbody>
</table>

(2 rows)

### \dj [ PATTERN ]

The `\dj` [ PATTERN ] meta-command returns all projections showing the schema, projection name, owner, and node. The returned rows include superprojections, live aggregate projections, Top-K projections, and projections with expressions:
If you supply a projection name as an argument, the system returns fewer records:

\djm call_center_dimension_n*

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Owner</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>online_sales</td>
<td>call_center_dimension_node0001</td>
<td>dbadmin</td>
<td>v_wmartdb_node0001</td>
</tr>
<tr>
<td>online_sales</td>
<td>call_center_dimension_node0002</td>
<td>dbadmin</td>
<td>v_wmartdb_node0002</td>
</tr>
<tr>
<td>online_sales</td>
<td>call_center_dimension_node0003</td>
<td>dbadmin</td>
<td>v_wmartdb_node0003</td>
</tr>
</tbody>
</table>

(3 rows)

\dn [ PATTERN ]

The `\dn [ PATTERN ]` meta-command returns the schema names and schema owner.

\djm

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_internal</td>
<td>dbadmin</td>
</tr>
<tr>
<td>v_catalog</td>
<td>dbadmin</td>
</tr>
<tr>
<td>v_monitor</td>
<td>dbadmin</td>
</tr>
<tr>
<td>public</td>
<td>dbadmin</td>
</tr>
<tr>
<td>store</td>
<td>dbadmin</td>
</tr>
<tr>
<td>online_sales</td>
<td>dbadmin</td>
</tr>
</tbody>
</table>

(6 rows)

The following command returns all schemas that begin with the letter v:

=> \djm v*

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_internal</td>
<td>dbadmin</td>
</tr>
<tr>
<td>v_catalog</td>
<td>dbadmin</td>
</tr>
<tr>
<td>v_monitor</td>
<td>dbadmin</td>
</tr>
</tbody>
</table>

(3 rows)
\dp [ PATTERN ]

The \dp [ PATTERN ] meta-command returns the grantee, grantor, privileges, schema, and name for all table access privileges in each schema:

```
vmartdb=> \dp
Access privileges for database "vmartdb"
Grantee    | Grantor  | Privileges | Schema | Name
---------   | -------- | ---------- | ------ | ----
  dbadmin   | USAGE    | public     | public |     
  dbadmin   | USAGE    | v_internal |       |     
  dbadmin   | USAGE    | v_catalog  |       |     
  dbadmin   | USAGE    | v_monitor  |       |     
(4 rows)
```

Note: \dp is the same as \z.

\ds [ PATTERN ]

The \ds [ PATTERN ] meta-command (lowercase s) returns a list of sequences and their parameters.

The following series of commands creates a sequence called my_seq and uses the vsql command to display its parameters:

```
=> CREATE SEQUENCE my_seq MAXVALUE 5000 START 150;
CREATE SEQUENCE

=> \ds
List of Sequences
Schema | Sequence | CurrentValue | IncrementBy | Minimum | Maximum | AllowCycle
------ | -------- | ------------ | ----------- | ------- | ------- | ----------
public | my_seq   | 149          | 1           | 1       | 5000    | f          
(1 row)
```

Note: You can return additional information about sequences by issuing SELECT * FROM SEQUENCES, as described in the SQL Reference Manual.

\dS [ PATTERN ]

The \dS [ PATTERN ] meta-command (uppercase S) returns all system table (monitoring API) names from the V_CATALOG and V_MONITOR schemas.

Tip: You can get identical results running this query: SELECT * FROM system_tables;
If you specify a schema name, you can view results for tables in that schema only; however, you must use the wildcard character. For example:

```plaintext
=> \dS v_catalog.*
```

You can also run the \dS command with a table argument to return information for that table only:

```plaintext
vmartdb=> \dS columns
    List of tables

    Schema   | Name   | Kind     | Description          | Comment
--------------------------------------------
    v_catalog | columns| system    | Table column information | |
(1 row)
```

If you provide the schema name with the table name, the output returns information about the table:

```plaintext
vmartdb=> \dS v_catalog.types
    List of tables

    Schema   | Name     | Kind     | Description                                         | Comment
---------------------------------------------
    v_catalog | types    | system    | Information about supported data types              | |
(1 row)
```

```plaintext\
\dt [ PATTERN ]
```

The \dt [ PATTERN ] meta-command (lowercase t) is identical to \d and returns all tables in the database—unless a table name is specified—in which case the command lists only the schema, name, kind and owner for the specified table (or tables if wildcards used).

```plaintext
vmartdb=> \dt inventory_fact
    List of tables

    Schema   | Name      | Kind     | Owner
-------------------------------------
    public   | inventory_fact | table | dbadmin
(1 row)
```

The following command returns all table names that begin with "st":

```plaintext
vmartdb=> \dt st*
    List of tables

    Schema   | Name     | Kind     | Owner
-------------------------------------
    store    | store_dimension | table | dbadmin
    store    | store_orders_fact | table | dbadmin
    store    | store_sales_fact | table | dbadmin
(3 rows)
```
\dT [ PATTERN ]

The \dT [ PATTERN ] meta-command (uppercase T) lists all supported data types.

\d@t
List of data types
type_name
-----------------------------------
Binary
Boolean
Char
Date
Float
Integer
Interval Day
Interval Day to Hour
Interval Day to Minute
Interval Day to Second
Interval Hour
Interval Hour to Minute
Interval Hour to Second
Interval Minute
Interval Minute to Second
Interval Month
Interval Second
Interval Year
Interval Year to Month
Long Varbinary
Long Varchar
Numeric
Time
TimeTz
Timestamp
TimestampTz
Uuid
Varbinary
Varchar
geography
geometry
(31 rows)

\dtv [ PATTERN ]

The \dtv [ PATTERN ] meta-command lists all tables and views, returning the schema, table or view name, kind (table or view), and owner.

vmartdb=> \dtv
List of tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Kind</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>online_sales</td>
<td>call_center_dimension</td>
<td>table</td>
<td>release</td>
</tr>
<tr>
<td>online_sales</td>
<td>online_page_dimension</td>
<td>table</td>
<td>release</td>
</tr>
</tbody>
</table>
\du [ PATTERN ]

The \du [ PATTERN ] meta-command returns all database users and attributes, such as if user is a superuser.

```
vmartdb=> \du
List of users
 User name | Is Superuser
----------+-------------------
dbadmin | t
(1 row)
```

\dv [ PATTERN ]

The \dv [ PATTERN ] meta-command returns the schema name, view name, and view owner.

The following example defines a view using the SEQUENCES system table:

```
vmartdb=> CREATE VIEW my_seqview AS (SELECT * FROM sequences);
CREATE VIEW

vmartdb=> \dv
List of views
 Schema | Name      | Owner
--------+-----------+-------
 public | my_seqview | dbadmin
(1 row)
```

If a view name is provided as an argument, the result shows the schema, view name, and the following for all columns within the view's result set: schema name, view name, column name, column data type, and data type size.
\echo

\e \edit [ FILE ]
\e \edit [ FILE ] edits the query buffer (or specified file) with an external editor. When the editor exits, its content is copied back to the query buffer. If no argument is given, the current query buffer is copied to a temporary file which is then edited in the same fashion.

The new query buffer is then re-parsed according to the normal rules of vsql, where the whole buffer up to the first semicolon is treated as a single line. (Thus you cannot make scripts this way. Use \i for that.) If there is no semicolon, vsql waits for one to be entered (it does not execute the query buffer).

Tip: vsql searches the environment variables VSQL_EDITOR, EDITOR, and VISUAL (in that order) for an editor to use. If all of them are unset, vi is used on Linux systems, notepad.exe on Windows systems.

\echo [ STRING ]
\echo [ STRING ] writes the string to standard output.

Tip: If you use the \o command to redirect your query output you might want to use \qecho instead of this command.
\f [ String ]
\f [ string ] sets the field separator for unaligned query output. The default is the vertical bar (|). See also \pset for a generic way of setting output options.

\g

The \g meta-command sends the query in the input buffer (see \p) to the server. With no arguments, it displays the results in the standard way.
\g FILE sends the query input buffer to the server, and writes the results to FILE.
\g COMMAND sends the query buffer to the server, and pipes the results to a shell COMMAND.

See Also

• \o

\H

\H toggles HTML query output format. This command is for compatibility and convenience, but see \pset about setting other output options.

\h \help

\h \help displays help information about the meta-commands. Works the same as \? .

=> \h
General
\[connect] [DBNAME|- [USER]]
connect to new database (currently "VMart")
\cd [DIR] change the current working directory
\q quit vsql
\set [NAME [VALUE]]
set internal variable, or list all if no parameters
\timing toggle timing of commands (currently off)
\unset NAME unset (delete) internal variable
\! [COMMAND] execute command in shell or start interactive shell
\password [USER]
<table>
<thead>
<tr>
<th>Query Buffer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>\e [FILE]</td>
<td>edit the query buffer (or file) with external editor</td>
</tr>
<tr>
<td>\g</td>
<td>send query buffer to server</td>
</tr>
<tr>
<td>\g [COMMAND]</td>
<td>send query buffer to server and pipe results to command</td>
</tr>
<tr>
<td>\p</td>
<td>show the contents of the query buffer</td>
</tr>
<tr>
<td>\r</td>
<td>reset (clear) the query buffer</td>
</tr>
<tr>
<td>\s [FILE]</td>
<td>display history or save it to file</td>
</tr>
<tr>
<td>\w FILE</td>
<td>write query buffer to file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>\echo [STRING]</td>
<td>write string to standard output</td>
</tr>
<tr>
<td>\i FILE</td>
<td>execute commands from file</td>
</tr>
<tr>
<td>\o [COMMAND]</td>
<td>pipe all query results to command</td>
</tr>
<tr>
<td>\o</td>
<td>close query-results file or pipe</td>
</tr>
<tr>
<td>\qecho [STRING]</td>
<td>write string to query output stream (see \o)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Informational</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d [PATTERN]</td>
<td>describe tables (list tables if no argument is supplied)</td>
</tr>
<tr>
<td>\df [PATTERN]</td>
<td>list functions</td>
</tr>
<tr>
<td>\dj</td>
<td>list projections</td>
</tr>
<tr>
<td>\dn</td>
<td>list schemas</td>
</tr>
<tr>
<td>\dp</td>
<td>list table access privileges</td>
</tr>
<tr>
<td>\ds</td>
<td>list sequences</td>
</tr>
<tr>
<td>\dT</td>
<td>list system tables</td>
</tr>
<tr>
<td>\dt [PATTERN]</td>
<td>list tables</td>
</tr>
<tr>
<td>\dtv [PATTERN]</td>
<td>list tables and views</td>
</tr>
<tr>
<td>\dT [PATTERN]</td>
<td>list data types</td>
</tr>
<tr>
<td>\du [PATTERN]</td>
<td>list users</td>
</tr>
<tr>
<td>\dv [PATTERN]</td>
<td>list views</td>
</tr>
<tr>
<td>\l</td>
<td>list all databases</td>
</tr>
<tr>
<td>\z [PATTERN]</td>
<td>list table access privileges (same as \dp)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formatting</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>toggle between unaligned and aligned output mode</td>
</tr>
<tr>
<td>\b</td>
<td>toggle beep on command completion</td>
</tr>
<tr>
<td>\C [STRING]</td>
<td>set table title, or unset if none</td>
</tr>
<tr>
<td>\f [STRING]</td>
<td>show or set field separator for unaligned query output</td>
</tr>
<tr>
<td>\H</td>
<td>toggle HTML output mode (currently off)</td>
</tr>
<tr>
<td>\pset NAME [VALUE]</td>
<td>set table output option</td>
</tr>
<tr>
<td></td>
<td>(NAME := {format</td>
</tr>
<tr>
<td></td>
<td>recordsep</td>
</tr>
<tr>
<td>\t</td>
<td>show only rows (currently off)</td>
</tr>
<tr>
<td>\T [STRING]</td>
<td>set HTML (&lt;table&gt; tag attributes, or unset if none</td>
</tr>
<tr>
<td>\x</td>
<td>toggle expanded output (currently off)</td>
</tr>
</tbody>
</table>

**\i FILE**

\i filename command reads input from the file `filename` and executes it as though it had been typed on the keyboard.

**Note:** To see the lines on the screen as they are read, set the variable `ECHO` to all.
Tip: The Vertica vsql client on Linux supports backquote (backtick) expansion. A simple example follows.

1. Set an environment variable to a path that contains scripts you want to run.

   ```bash
   $ export MYSCRIPTS=/home/dbadmin/testscripts
   ```

2. Issue the vsql command.

   ```bash
   $ vsql
   ```

3. Use backquote expansion to include the path for running an existing script (for example, sample.sql).

   ```bash
   => \i `echo $MYSCRIPTS/sample.sql`
   ```

\`

\` provides a list of databases and their owners.

```bash
vmartdb=> \l
List of databases
  name  | user_name
---------+------------
  vmartdb | dbadmin
(1 row)
```

\`locale\`

The vsql \`locale\` command displays the current locale setting or lets you set a new locale for the session.

This command does not alter the default locale for all database sessions. To change the default for all sessions, set the `DefaultSessionLocale` configuration parameter.

**Viewing the Current Locale Setting**

To view the current locale setting, use the vsql command \`locale\`, as follows:

```bash
=> \locale
en_US@collation=binary
```
Overriding the Default Local for a Session

To override the default local for a specific session, use the vsql command `\locale <ICU-locale-identifier>`. The session locale setting applies to any subsequent commands issued in the session.

For example:

```plaintext
=> \locale en_GBINFO:
INFO 2567: Canonical locale: 'en_GBINFO:'
Standard collation: 'LEN'
English (GBINFO:)
```

Notes

The server locale settings impact only the collation behavior for server-side query processing. The client application is responsible for ensuring that the correct locale is set in order to display the characters correctly. Below are the best practices recommended by Vertica to ensure predictable results:

- The locale setting in the terminal emulator for vsql (POSIX) should be set to be equivalent to session locale setting on server side (ICU) so data is collated correctly on the server and displayed correctly on the client.

- The vsql locale should be set using the POSIX LANG environment variable in terminal emulator. Refer to the documentation of your terminal emulator for how to set locale.

- Server session locale should be set using the set as described in Specify the Default Locale for the Database.

- Note that all input data for vsql should be in UTF-8 and all output data is encoded in UTF-8.

- Non UTF-8 encodings and associated locale values are not supported.

\o

The \o meta-command is used to control where vsql directs its query output. The output can be written to a file, piped to a shell command, or sent to the standard output.

- \o FILE sends all subsequent query output to FILE.
- \o COMMAND pipes all subsequent query output to a shell COMMAND.
\o with no argument closes any open file or pipe, and switches back to normal query result output.

Notes

- Query results includes all tables, command responses, and notices obtained from the database server.
- To intersperse text output with query results, use \qecho.

See Also

- \g

\p

\p prints the current query buffer to the standard output. For example:

```
=> \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);
```

\password [ USER ]

\password starts the password change process. Users can only change their own passwords. The command prompts the user for the old password, a new password, and then the new password again to confirm.

A superuser can change the password of another user by supplying the username. A superuser is not prompted for the old password, either when changing his or her own password, or when changing another user’s password.

Note: If you want to cancel the password change process, press ENTER until you return the to vsqi prompt.
\pset NAME [ VALUE ]

\pset NAME [ VALUE ] sets options affecting the output of query result tables. NAME describes which option to set, as illustrated in the following table. The parameters of VALUE depend thereon.

It is an error to call \pset without arguments.

Adjustable printing options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>format</strong></td>
<td>Sets the output format to one of unaligned, aligned, html, or \LaTeX. Unique abbreviations are allowed. (That would mean one letter is enough.) &quot;Unaligned&quot; writes all columns of a row on a line, separated by the currently active field separator. This is intended to create output that might be intended to be read in by other programs (tab-separated, comma-separated). &quot;Aligned&quot; mode is the standard, human-readable, nicely formatted text output that is default. The &quot;HTML&quot; and &quot;LaTeX&quot; modes put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)</td>
</tr>
<tr>
<td><strong>border</strong></td>
<td>The second argument must be a number. In general, the higher the number the more borders and lines the tables have, but this depends on the particular format. In HTML mode, this translates directly into the \texttt{border=...} attribute, in the others only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.</td>
</tr>
<tr>
<td><strong>expanded</strong></td>
<td>Toggles between regular and expanded format. When expanded format is enabled, all output has two columns with the column name on the left and the data on the right. This mode is useful if the data wouldn't fit on the screen in the normal &quot;horizontal&quot; mode. Expanded mode is supported by all four output formats. \texttt{x} is the same as \texttt{\pset expanded}.</td>
</tr>
<tr>
<td><strong>fieldsep</strong></td>
<td>Specifies the field separator to be used in unaligned output mode. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type \pset fieldsep 't'. The default field separator is '</td>
</tr>
<tr>
<td><strong>footer</strong></td>
<td>Toggles the display of the default footer (x rows).</td>
</tr>
<tr>
<td><strong>null</strong></td>
<td>The second argument is a string that is printed whenever a column is null. The default is not to print anything, which can easily be mistaken for, say, an empty string. Thus, one might choose to write \pset null '(null)'.</td>
</tr>
<tr>
<td><strong>recordsep</strong></td>
<td>Specifies the record (line) separator to use in unaligned output mode. The default is a newline character.</td>
</tr>
<tr>
<td><strong>trailingrecordsep</strong></td>
<td>Toggles on or off the trailing record separator to use in unaligned output mode.</td>
</tr>
<tr>
<td><strong>tuples_only (or t)</strong></td>
<td>Toggles between tuples only and full display. Full display might show extra information such as column headers, titles, and various footers. In tuples only mode, only actual table data is shown.</td>
</tr>
<tr>
<td><strong>title [ text ]</strong></td>
<td>Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no argument is given, the title is unset.</td>
</tr>
<tr>
<td><strong>tableattr (or T)[ text ]</strong></td>
<td>Allows you to specify any attributes to be placed inside the HTML table tag. This could for example be cellpadding or bgcolor. Note that you probably don't want to specify border here, as that is already taken care of by \pset border.</td>
</tr>
<tr>
<td><strong>pager</strong></td>
<td>Controls use of a pager for query and vsql help output. If the environment variable PAGER is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as more) is used. When the pager is off, the pager is not used. When the pager is on, the pager is used only when appropriate; that is, the output is to a terminal and does not fit on the screen. (vsql does not do a perfect job of estimating when to use the pager.) \pset pager turns the pager on and off. Pager can also be set to always, which causes the pager to be always used.</td>
</tr>
</tbody>
</table>
See illustrations on how these different formats look in the Examples section.

Tip: There are various shortcut commands for \pset. See \a, \C, \H, \t, \T, and \x.

\q
\q quits the vsql program.

\qecho [ STRING ]
\qecho [ STRING ] is identical to \echo except that the output is written to the query output stream, as set by \o.

\r
\r resets (clears) the query buffer.
For example, run the \p meta-command to see what is in the query buffer:

=&gt; \p
CREATE VIEW my_seqview AS (SELECT * FROM sequences);

Now reset the query buffer:

=&gt; \r
Query buffer reset (cleared).

If you reissue the command to see what's in the query buffer, you can see it is now empty:

=&gt; \p
Query buffer is empty.

\s [ FILE ]
\s [ FILE ] prints or saves the command line history to filename. If a filename is not specified, \s writes the history to the standard output. This option is only available if vsql is configured to use the GNU Readline library.
\set [ NAME [ VALUE [ ... ] ] ]

\set [ name [ value [ ... ] ] ] sets the internal variable name to value or, if more than one value is given, to the concatenation of all of values. If no second argument is given, the variable is set with no value.

If no argument is provided, \set lists all internal variables; for example:

```
=> \set
VERSION = 'Vertica Analytic Database v6.0.0-0'
AUTOCOMMIT = 'off'
VERBOSITY = 'default'
PROMPT1 = '%/%R%#'
PROMPT2 = '%/%R%#'
PROMPT3 = '>>'
ROWS_AT_A_TIME = '1000'
DBNAME = 'VMartDB'
USER = 'dbadmin'
HOST = '<host_ip_address>'
PORT = '5433'
LOCALE = 'en_US@collation=binary'
HISTSIZE = '500'
```

Notes

- Valid variable names are case sensitive and can contain characters, digits, and underscores. vsql treats several variables as special, which are described in Variables.

- The \set parameter ROWS_AT_A_TIME defaults to 1000. It retrieves results as blocks of rows of that size. The column formatting for the first block is used for all blocks, so in later blocks some entries could overflow. See \timing for examples.

- When formatting results, Vertica buffers ROWS_AT_A_TIME rows in memory to calculate the maximum column widths. It is possible that rows after this initial fetch are not properly aligned if any of the field values are longer than those see in the first ROWS_AT_A_TIME rows. ROWS_AT_A_TIME can be \unset to guarantee perfect alignment, but this requires re-buffering the entire result set in memory and may cause vsql to fail if the result set is too big.

- To unset a variable, use the \unset command.
Using Backquotes to Read System Variables

In vsql, the contents of backquotes are passed to the system shell to be interpreted (the same behavior as many UNIX shells). This is particularly useful in setting internal vsql variables, since you may want to access UNIX system variables (such as HOME or TMPDIR) rather than hard-code values.

For example, if you want to set an internal variable to the full path for a file in your UNIX user directory, you could use backquotes to get the content of the system HOME variable, which is the full path to your user directory:

```
=> \set inputfile `echo $HOME`/myinput.txt
  \echo :inputfile
/home/dbadmin/myinput.txt
```

The contents of the backquotes are replaced with the results of running the contents in a system shell interpreter. In this case, the `echo $HOME` command returns the contents of the HOME system variable.

\t

\t toggles the display of output column name headings and row count footer. This command is equivalent to `\pset tuples_only` and is provided for convenience.

\T [ STRING ]

\T [ STRING ] specifies attributes to be placed within the table tag in HTML tabular output mode. This command is equivalent to `\pset tableattr table_options`.

\timing

This meta-command reports, in milliseconds, the length of time each SQL statement runs. The results include:

- the length of time required to fetch the first block of rows
- the total time until the last block is formatted

For example:
=> SELECT * from allsales;

state | name | sales
------|------|-------
NY    | B    | 20
NY    | C    | 15
MA    | A    | 40

(3 rows)

Time: First fetch (7 rows): 59.264 ms. All rows formatted: 59.362 ms

Timing is off by default. To determine the state of the timing function, enter the following:

=> \timing

Timing is on

This example indicates that timing is off.

**Toggle \timing function**

You can set the timing state by entering on or off:

=> \timing on

Timing is on

=> \timing off

Timing is off

Alternatively, enter the function name without an argument to automatically toggle the state, whether it's on or off. For example, if you run \timing on, running \timing disables timing:

=> \timing on

Timing is on

=> \timing

Timing is off

**Enable \timing from the vsql Command Line**

You can enable \timing from the command line using the vsql -i command. You can only use -i with the -c (command) and -f (filename) commands. For more information see Command-Line Options.

From the command line enter the -i option before running a session to turn timing on. For example:
Example

The following example shows a SQL command with timing on:

```sql
=> \SELECT user_name, ssl_state, authentication_method, client_authentication_name, client_type FROM sessions
   WHERE session_id=(SELECT session_id FROM current_session);
```

```
+---------+--------+-------------------+-----------------+-------+
| user_name | ssl_state | authentication_method | client_authentication_name | client_type |
|-----------+----------+----------------------+-----------------------------+------------|
| dbadmin   | None     | ImpTrust             | default; Implicit Trust     | vsql       |
| (1 row)   |          |                      |                             |            |
```

The following table provides a quick guide on what occurs when you run \timing commands starting with the \timing being on:

<table>
<thead>
<tr>
<th>vsql command</th>
<th>Starting \timing state = on</th>
</tr>
</thead>
<tbody>
<tr>
<td>=&gt; \timing on</td>
<td>Keeps timing on</td>
</tr>
<tr>
<td>=&gt; \timing</td>
<td>Turns timing on</td>
</tr>
<tr>
<td>=&gt; \timing off</td>
<td>Turns timing off</td>
</tr>
<tr>
<td>=&gt; \timing on</td>
<td>Turns timing on</td>
</tr>
</tbody>
</table>

\unset [ NAME ]

\unset [ NAME ] unsets (deletes) the internal variable name that was set using the \set meta-command.

\w [ FILE ]

\w [ FILE ] outputs the current query buffer to the file filename.
\x

\x toggles extended table formatting mode. Is equivalent to \pset expanded.

Note: There is no space between the backslash and the x.

\z

\z lists table access privileges (grantee, grantor, privilege, and name) for all table access privileges in each schema. Is the same as \dp .
Variables

vsql provides variable substitution features similar to common Linux command shells. Variables are simply name/value pairs, where the value can be any string of any length. To set variables, use the vsql meta-command \set:

```plaintext
=> \set fact dim
```

sets the variable fact to the value dim. To retrieve the content of the variable, precede the name with a colon and use it as the argument of any slash command:

```plaintext
=> \echo :fact dim
```

Note: The arguments of \set are subject to the same substitution rules as with other commands. For example, \set dim :fact is a valid way to copy a variable.

If you call \set without a second argument, the variable is set, with an empty string as value. To unset (or delete) a variable, use the command \unset.

vsql's internal variable names can consist of letters, numbers, and underscores in any order and any number. Some of these variables are treated specially by vsql. They indicate certain option settings that can be changed at run time by altering the value of the variable or represent some state of the application. Although you can use these variables for any other purpose, this is not recommended. By convention, all specially treated variables consist of all upper-case letters (and possibly numbers and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes.

SQL Interpolation

An additional useful feature of vsql variables is that you can substitute ("interpolate") them into regular SQL statements. The syntax for this is again to prepend the variable name with a colon (:).

```plaintext
=> \set fact 'my_table'
=> SELECT * FROM :fact;
```

would then query the table my_table. The value of the variable is copied literally (except for backquoted strings, see below), so it can even contain unbalanced quotes or backslash commands. Make sure that it makes sense where you put it. Variable interpolation is not performed into quoted SQL entities.
AUTOCOMMIT

When AUTOCOMMIT is set 'on', each SQL command is automatically committed upon successful completion; for example:

```
\set AUTOCOMMIT on
```

To postpone COMMIT in this mode, set the value as off.

```
\set AUTOCOMMIT off
```

If AUTOCOMMIT is empty or defined as off, SQL commands are not committed unless you explicitly issue COMMIT.

Notes

- AUTOCOMMIT is off by default.

- AUTOCOMMIT must be in uppercase, but the values, on or off, are case insensitive.

- In autocommit-off mode, you must explicitly abandon any failed transaction by entering ABORT or ROLLBACK.

- If you exit the session without committing, your work is rolled back.

- Validation on vsq1 variables is done when they are run, not when they are set.

- The COPY statement, by default, commits on completion, so it does not matter which AUTOCOMMIT mode you use, unless you issue COPY NO COMMIT. Please note that DDL statements are autocommitted.

- To tell if AUTOCOMMIT is on or off, issue the set command:

```
$ \set...
  AUTOCOMMIT = 'off'
  ...
```
- **AUTOCOMMIT** is off if a `SELECT * FROM LOCKS` shows locks from the statement you just ran.

```sql
$ \set AUTOCOMMIT off
$ \set ...
  AUTOCOMMIT = 'off'
...
SELECT COUNT(*) FROM customer_dimension;
count
-------
50000
(1 row)
SELECT node_names, object_name, lock_mode, lock_scope
FROM LOCKS;
 node_names | object_name | lock_mode | lock_scope
-----------------------------------------------
site01    | Table:customer_dimension | S         | TRANSACTION
(1 row)
```

**DBNAME**

The name of the database to which you are currently connected. **DBNAME** is set every time you connect to a database (including program startup), but it can be unset.

**ECHO**

If set to all, all lines entered from the keyboard or from a script are written to the standard output before they are parsed or run.

To select this behavior on program start-up, use the switch `-a`. If set to `queries`, `vsql` merely prints all queries as they are sent to the server. The switch for this is `-e`.

**ECHO_HIDDEN**

When this variable is set and a backslash command queries the database, the query is first shown. This way you can study the Vertica internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E`.)

If you set the variable to the value `noexec`, the queries are just shown but are not actually sent to the server and run.
ENCODING

The current client character set encoding.

HISTCONTROL

If this variable is set to ignore space, lines that begin with a space are not entered into the history list. If set to a value of ignoredups, lines matching the previous history line are not entered. A value of ignoreboth combines the two options. If unset, or if set to any other value than those previously mentioned, all lines read in interactive mode are saved on the history list.

Source: Bash.

HISTSIZE

The number of commands to store in the command history. The default value is 500.

Source: Bash.

HOST

The database server host you are currently connected to. This is set every time you connect to a database (including program startup), but can be unset.

IGNOREEEOF

If unset, sending an EOF character (usually Control+D) to an interactive session of vsq1 terminates the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

Source: Bash.
ON_ERROR_STOP

By default, if a script command results in an error, for example, because of a malformed command or invalid data format, processing continues. If you set ON_ERROR_STOP to ‘on’ in a script and an error occurs during processing, the script terminates immediately.

If you set ON_ERROR_STOP to ‘on’ in a script, run the script from Linux using vsql -f <filename>, and an error occurs, vsql returns an error code 3 to Linux to indicate that the error occurred in a script.

To enable ON_ERROR_STOP:

```sql
=> \set ON_ERROR_STOP on
```

To disable ON_ERROR_STOP:

```sql
=> \set ON_ERROR_STOP off
```

PORT

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

PROMPT1 PROMPT2 PROMPT3

These specify what the prompts vsql issues look like. See Prompting for details.

QUIET

This variable is equivalent to the command line option `-q`. It is probably not too useful in interactive mode.

SINGLELINE

This variable is equivalent to the command line option `-S`. 
SINGLESTEP

This variable is equivalent to the command line option `-s`.

USER

The database user you are currently connected as. This is set every time you connect to a database (including program startup), but can be unset.

VERBOSITY

This variable can be set to the values `default`, `verbose`, or `terse` to control the verbosity of error reports.

VSQL_HOME

By default, the vsql program reads configuration files from the user's home directory. In cases where this is not desirable, the configuration file location can be overridden by setting the VSQL_HOME environment variable in a way that does not require modifying a shared resource.

In the following example, vsql reads configuration information out of `/tmp/jsmith` rather than out of `~`.

```
# Make an alternate configuration file in /tmp/jsmith
mkdir -p /tmp/jsmith
echo "\echo Using VSQLRC in tmp/jsmith" > /tmp/jsmith/.vsqlrc
# Note that nothing is echoed when invoked normally
vsql
# Note that the .vsqlrc is read and the following is
displayed before the vsql prompt
#
# Using VSQLRC in tmp/jsmith
VSQL_HOME=/tmp/jsmith vsql
```

VSQL_SSLMODE

VSQL_SSLMODE specifies how (or whether) clients (like admintools) use SSL when connecting to servers. The default value is `prefer`, meaning to use SSL if the server offers it. Legal values
are require, prefer, allow, and disable. This variable is equivalent to the command-line
-m option (or -s -sslmode).

**Prompting**

The prompts vsql issues can be customized to your preference. The three variables PROMPT1, PROMPT2, and PROMPT3 contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when vsql requests a new command. Prompt 2 is issued when more input is expected during command input because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you run a SQL COPY command and you are expected to type in the row values on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (%) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

<table>
<thead>
<tr>
<th>Substitute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%M</td>
<td>The full host name (with domain name) of the database server, or [local] if the connection is over a socket, or [local:/dir/name], if the socket is not at the compiled in default location.</td>
</tr>
<tr>
<td>%m</td>
<td>The host name of the database server, truncated at the first dot, or [local].</td>
</tr>
<tr>
<td>%&gt;</td>
<td>The port number at which the database server is listening.</td>
</tr>
<tr>
<td>%n</td>
<td>The database session user name.</td>
</tr>
<tr>
<td>%/</td>
<td>The name of the current database.</td>
</tr>
<tr>
<td>%~</td>
<td>Like %/, but the output is ~ (tilde) if the database is your default database.</td>
</tr>
<tr>
<td>%#</td>
<td>If the session user is a database superuser, then a #, otherwise a &gt;. (The expansion of this value might change during a database session as the result of the command SET SESSION AUTHORIZATION.)</td>
</tr>
<tr>
<td>%R</td>
<td>In prompt 1 normally =, but ^ if in single-line mode, and ! if the session is disconnected from the database (which can happen if \connect fails). In prompt 2 the sequence is replaced by -, <em>, a single quote, a double quote, or a dollar sign, depending on whether vsql expects more input because the command wasn't terminated yet, because you are inside a /</em> ... */ comment, or because you are inside a quoted or dollar-escaped string. In prompt 3 the sequence doesn't produce anything.</td>
</tr>
</tbody>
</table>
Transaction status: an empty string when not in a transaction block, or * when in a transaction block, or ! when in a failed transaction block, or ? when the transaction state is indeterminate (for example, because there is no connection).

The character with the indicated numeric code is substituted. If digits starts with 0x the rest of the characters are interpreted as hexadecimal; otherwise if the first digit is 0 the digits are interpreted as octal; otherwise the digits are read as a decimal number.

The value of the vsql variable name. See the section Variables for details.

The output of command, similar to ordinary "back-tick" substitution.

Prompts may contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for the line editing features of Readline to work properly, these non-printing control characters must be designated as invisible by surrounding them with %[ and %]. Multiple pairs of these may occur within the prompt. The following example results in a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible, color-capable terminals:

testdb=> set PROMPT1 '%[033[1;33;40m%]n@%/%R%#%[033[0m%]'

To insert a percent sign into your prompt, write %%. The default prompts are '%/%R#' for prompts 1 and 2, and '>>' for prompt 3.

Note: See the specification for terminal control sequences (applicable to gnome-terminal and xterm).

Command Line Editing

vsql supports the tecla library for convenient line editing and retrieval.

The command history is automatically saved when vsql exits and is reloaded when vsql starts up. Tab-completion is also supported, although the completion logic makes no claim to be a SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named .teclarc in your home directory:

bind ^I

Read the tecla documentation for further details.
Notes

The vsql implementation of the tecla library deviates from the tecla documentation as follows:

- **Recalling Previously Typed Lines**

  Under pure tecla, all new lines are appended to a list of historical input lines maintained within the GetLine resource object. In vsql, only different, non-empty lines are appended to the list of historical input lines.

- **History Files**

  tecla has no standard name for the history file. In vsql, the file name is called ~/.vsql_hist.

- **International Character Sets (Meta keys and locales)**

  In vsql, 8-bit meta characters are no longer supported. Make sure that meta characters send an escape by setting their EightBitInput X resource to False. You can do this in one of the following ways:

  - Edit the ~/.Xdefaults file by adding the following line:

    ```
    XTerm*EightBitInput: False
    ```

  - Start an xterm with an -xrm '*EightBitInput: False' command-line argument.

- **Key Bindings:**

  The following key bindings are specific to vsql:

  - *Insert* switches between insert mode (the default) and overwrite mode.
  - *Delete* deletes the character to the right of the cursor.
  - *Home* moves the cursor to the front of the line.
  - *End* moves the cursor to the end of the line.
  - ^R Performs a history backwards search.
# vsql Environment Variables

Set one or more of the following environment variables to be used by the defined properties automatically, each time you start vsql:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGER</td>
<td>If the query results do not fit on the screen, they are piped through this command. Typical values are more or less. The default is platform-dependent. The use of the pager can be disabled by using the \pset command.</td>
</tr>
<tr>
<td>TMPDIR</td>
<td>Directory for storing temporary files. The default is platform-dependent. On Unix-like systems the default is /tmp.</td>
</tr>
<tr>
<td>VSQL_EDITOR</td>
<td>Editor used by the \e command. The variables are examined in the order listed; the first that is set is used.</td>
</tr>
<tr>
<td>VSQL_HOME</td>
<td>By default, the vsql program reads configuration files from the user's home directory. In cases where this is not desirable, the configuration file location can be overridden by setting the VSQL_HOME environment variable in a way that does not require modifying a shared resource.</td>
</tr>
<tr>
<td>VSQL_HOST</td>
<td>Host name or IP address of the Vertica node.</td>
</tr>
<tr>
<td>VSQL_PASSWORD</td>
<td>The database password. Using this environment variable increases site security by precluding the need to enter the database password on the command line.</td>
</tr>
<tr>
<td>VSQL_PORT</td>
<td>Port to use for the connection.</td>
</tr>
<tr>
<td>VSQL_SSLMODE</td>
<td>Specifies whether and how clients such as admintools use SSL when connecting to servers.</td>
</tr>
<tr>
<td>VSQL_USER</td>
<td>User name to use for the connection.</td>
</tr>
</tbody>
</table>
**Locales**

The default terminal emulator under Linux is gnome-terminal, although xterm can also be used.

Vertica recommends that you use gnome-terminal with vsq1 in UTF-8 mode, which is its default.

**To Change Settings on Linux**

1. From the tabs at the top of the vsq1 screen, select Terminal.
2. Click Set Character Encoding.
3. Select Unicode (UTF-8).

**Note:** This works well for standard keyboards. xterm has a similar UTF-8 option.

**To Change Settings on Windows Using PuTTy**

1. Right click the vsq1 screen title bar and select Change Settings.
2. Click Window and click Translation.
3. Select UTF-8 in the drop-down menu on the right.

**Notes**

- vsq1 has no way of knowing how you have set your terminal emulator options.

- The tecla library is prepared to do POSIX-type translations from a local encoding to UTF-8 on interactive input, using the POSIX LANG, etc., environment variables. This could be useful to international users who have a non-UTF-8 keyboard. See the tecla documentation for details.
Vertica recommends the following (or whatever other .UTF-8 locale setting you find appropriate):

```bash
export LANG=en_US.UTF-8
```

- The `vsql \locale` command invokes and tracks the server `SET LOCALE TO` command, described in the SQL Reference Manual. `vsql` itself currently does nothing with this locale setting, but rather treats its input (from files or from tecla), all its output, and all its interactions with the server as UTF-8. `vsql` ignores the POSIX locale variables, except for any "automatic" uses in `printf`, and so on.

### Entering Data with `vsql`

You often need to insert literal data when using `vsql`. For example:

- Adding a row of data to a table using an `INSERT` statement.

- Adding multiple rows of data through a `COPY FROM STDIN` statement.

The following table lists the data types that Vertica supports, and the format you use to enter that data in queries when using `vsql`.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Inserting to <code>vsql</code> using</th>
<th>Example Use in <code>INSERT INTO table...</code></th>
<th>For More Information See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary types, such as BINARY and VARBINARY</td>
<td>Helper functions such as HEX_TO_BINARY, octal strings, specified data format in COPY statements, casting string values to binary.</td>
<td>VALUES(HEX_TO_BINARY ('\0x3D'), '\141\337');</td>
<td>• Binary Data Types</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Loading Binary (Native) Data</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Literal values TRUE and FALSE or strings such as 'y', 't', 'true', or &quot;f&quot;</td>
<td>VALUES(TRUE, 'f');</td>
<td>Boolean Data Type</td>
</tr>
</tbody>
</table>

Vertica Analytic Database (9.0.x) Page 4436 of 6180
'false'.

<table>
<thead>
<tr>
<th>Character data types such as CHAR or LONG VARCHAR</th>
<th>Strings enclosed in single quotes.</th>
<th>VALUES('my string');</th>
<th>Character Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date and time data types, such as TIMESTAMPTZ</td>
<td>Formatted text string</td>
<td>VALUES('16:43:00', '2016-09-15 04:55:00 PDT');</td>
<td>Date/Time Data Types</td>
</tr>
<tr>
<td>Date/Time Data Types</td>
<td>Date/Time Expressions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Data Types</td>
<td>Literal numeric values, including scientific notation, hexadecimal, and BINARY scaling.</td>
<td>VALUES(3.1415, 42, 6.0221409e23);</td>
<td>Numeric Data Types</td>
</tr>
<tr>
<td>UUID</td>
<td>Formatted text string</td>
<td>VALUES('12345678-1234-1234-1234-123456789012');</td>
<td>UUID Data Type</td>
</tr>
</tbody>
</table>

**Files**

Before starting up, vsq1 attempts to read and execute commands from the system-wide vsq1rc file and the user's ~/.vsq1rc file. The command-line history is stored in the file ~/.vsq1_history.

**Tip:** If you want to save your old history file, open another terminal window and save a copy to a different file name.

**Exporting Data Using vsq1**

You can use vsq1 for simple data-export tasks by changing its output format options so the output is suitable for importing into other systems (tab-delimited or comma-separated files, for example). These options can be set either from within an interactive vsq1 session, or through command-line arguments to the vsq1 command (making the export process suitable for
automation through scripting). After you have set vsq1’s options so it outputs the data in a format your target system can read, you run a query and capture the result in a text file.

The following table lists the meta-commands and command-line options that are useful for changing the format of vsq1’s output.

<table>
<thead>
<tr>
<th>Description</th>
<th>Meta-command</th>
<th>Command-line Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable padding used to align output.</td>
<td>\a</td>
<td>-A or --no-align</td>
</tr>
<tr>
<td>Show only tuples, disabling column headings and row counts.</td>
<td>\t</td>
<td>-t or --tuples-only</td>
</tr>
<tr>
<td>Set the field separator character.</td>
<td>\pset fieldsep</td>
<td>-F or --field-separator</td>
</tr>
<tr>
<td>Send output to a file.</td>
<td>\o</td>
<td>-o or --output</td>
</tr>
<tr>
<td>Specify a SQL statement to execute.</td>
<td>N/A</td>
<td>-c or --command</td>
</tr>
</tbody>
</table>

The following example demonstrates disabling padding and column headers in the output, and setting a field separator to dump a table to a tab-separated text file within an interactive session.

```sql
=> SELECT * FROM my_table;
 a | b | c
---+---+---
 a | one | 1
 b | two | 2
 c | three | 3
 d | four | 4
 e | five | 5
(5 rows)
=> \a
Output format is unaligned.
=> \t
Showing only tuples.
=> \pset fieldsep \t
Field separator is "	".
=> \o dumpfile.txt
=> select * from my_table;
=> \o
=> \! cat dumpfile.txt
 a   one   1
 b   two   2
 c   three | 3
 d   four   4
 e   five   5
```
**Note:** You could encounter issues with empty strings being converted to NULLs or the reverse using this technique. You can prevent any confusion by explicitly setting null values to output a unique string such as NULLNULLNULL (for example, \pset null 'NULLNULLNULL '). Then, on the import end, convert the unique string back to a null value. For example, if you are copying the file back into a Vertica database, you would give the argument NULL 'NULLNULLNULL' to the COPY statement.

When logged into one of the database nodes, you can create the same output file directly from the command line by passing the right parameters to vsql:

```
$ vsql -U username -F $'\t' -At -o dumpfile.txt -c "SELECT * FROM my_table;"
Password:
$ cat dumpfile.txt
a  one  1
b  two  2
c  three 3
d  four 4
e  five 5
```

If you want to convert null values to a unique string as mentioned earlier, you can add the argument `-P null='NULLNULLNULL'` (or whatever unique string you choose).

By adding the `-w` vsql command-line option to the example command line, you could use the command within a batch script to automate the data export. However, the script would contain the database password as plain text. If you take this approach, you should prevent unauthorized access to the batch script, and also have the script use a database user account that has limited access.

To set the field separator value to a control character, use your shell's control character escape notation. In Bash, you specify a control character in an argument using a dollar sign ($) followed by a string contained in single quotes. This string can contain C-string escapes (such as \t for tab), or a backslash (\) followed by an octal value for the character you want to use.

The following example demonstrates setting the separator character to tab (\t), vertical tab (\v) and the octal value of vertical tab (\013).

```
$ vsql -At -c "SELECT * FROM testtable;"
A|1|2|3
B|4|5|6

$ vsql -F $'\t' -At -c "SELECT * FROM testtable;"
A  1  2  3
B  4  5  6

$ vsql -F $'\v' -At -c "SELECT * FROM testtable;"
A 1
  2
  3
```
Copying Data Using vsq1

You can use vsq1 to copy data between two Vertica databases. This technique is similar to the technique explained in Exporting Data Using vsq1, except instead of having vsq1 save data to a file for export, you pipe one vsq1's output to the input of another vsq1 command that runs a COPY statement from STDIN. This technique can also work for other databases or applications that accept data from an input stream.

Note: The following technique only works for individual tables. To copy an entire database to another cluster, see Copying the Database to Another Cluster in the Administrator's Guide.

The easiest way to copy using vsq1 is to log in to a node of the target database, then issue a vsq1 command that connects to the source Vertica database to dump the data you want. For example, the following command copies the store.store_sales_fact table from the vmart database on node testdb01 to the vmart database on the node you are logged into:

```bash
vsq1 -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \ 
| vsq1 -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

Note: The above example copies the data only, not the table design. The target table for the data copy must already exist in the target database. You can export the design of the table using EXPORT_OBJECTS or EXPORT_CATALOG.

If you are using the Bash shell, you can escape special delimiter characters. For example, DELIMITER E \t specifies tab. Shells other than Bash may have other string-literal syntax.
Monitoring Progress (optional)

You may want some way of monitoring progress when copying large amounts of data between Vertica databases. One way of monitoring the progress of the copy operation is to use a utility such as Pipe Viewer that pipes its input directly to its output while displaying the amount and speed of data it passes along. Pipe Viewer can even display a progress bar if you give it the total number of bytes or lines you expect to be processed. You can get the number of lines to be processed by running a separate vsql command that executes a SELECT COUNT query.

**Note:** Pipe Viewer isn't a standard Linux or Solaris command, so you will need to download and install it yourself. See the Pipe Viewer page for download packages and instructions. Vertica does not support Pipe Viewer. Install and use it at your own risk.

The following command demonstrates how you can use Pipe Viewer to monitor the progress of the copy shown in the prior example. The command is complicated by the need to get the number of rows that will be copied, which is done using a separate vsql command within a Bash backquote string, which executes the string's contents and inserts the output of the command into the command line. This vsql command just counts the number of rows in the store.store_sales_fact table.

```
vsq1 -U username -w passwd -h testdb01 -d vmart -At -c "SELECT * from store.store_sales_fact" \ 
| pv -lpetr -s `vsq1 -U username -w passwd -h testdb01 -d vmart -At -c "SELECT COUNT (*) FROM store.store_sales_fact;"` \ 
| vsq1 -U username -w passwd -d vmart -c "COPY store.store_sales_fact FROM STDIN DELIMITER '|';"
```

While running, the above command displays a progress bar that looks like this:

```
0:00:39 [12.6M/s] [==================================] 50% ETA 00:00:40
```

Output Formatting Examples

The first example shows how to spread a command over several lines of input. Notice the changing prompt:

```
=> CREATE TABLE my_table (  
-> first integer not null default 0,  
-> second char(10));
CREATE TABLE
```

Assume you have filled the table with data and want to take a look at it:
testdb=> SELECT * FROM my_table;
first | second
--------+--------
 1 | one
 2 | two
 3 | three
 4 | four
(4 rows)

You can display tables in different ways by using the \pset command:

```
testdb=> \pset border 2
Border style is 2.
testdb=> SELECT * FROM my_table;
+-------+-------+
| first  | second|
+-------+-------+
| 1     | one   |
| 2     | two   |
| 3     | three |
| 4     | four  |
+-------+-------+
(4 rows)
=> \pset border 0
Border style is 0.
=> SELECT * FROM my_table;
first second
------ ------
 1 one
 2 two
 3 three
 4 four
(4 rows)
=> \pset border 1 Border style is 1.
=> \pset format unaligned
Output format is unaligned.
=> \pset fieldsep ','
Field separator is ",".
=> \pset tuples_only
Showing only tuples.
=> SELECT second, first FROM my_table;
one,1
two,2
three,3
four,4
```

Alternatively, use the short commands:

```
=> \a \t x Output format is aligned.
Tuples only is off.
Expanded display is on.
=> SELECT * FROM my_table;
first | 1
second | one
--------+-----
first | 2
```
<table>
<thead>
<tr>
<th>second</th>
<th>two</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>3</td>
</tr>
<tr>
<td>second</td>
<td>three</td>
</tr>
<tr>
<td>first</td>
<td>4</td>
</tr>
<tr>
<td>second</td>
<td>four</td>
</tr>
</tbody>
</table>
Client Libraries

The Vertica client driver libraries provide interfaces for connecting your client applications (or third-party applications such as Cognos and MicroStrategy) to your Vertica database. The drivers simplify exchanging data for loading, report generation, and other common database tasks.

There are three separate client drivers:

- Open Database Connectivity (ODBC)—the most commonly-used interface for third-party applications and clients written in C, Python, PHP, Perl, and most other languages.
- Java Database Connectivity (JDBC)—used by clients written in the Java programming language.
- ActiveX Data Objects for .NET (ADO.NET)—used by clients developed using Microsoft's .NET Framework and written in C#, Visual Basic .NET, and other .NET languages.

Client Driver Standards

The Vertica client drivers are compatible with the following driver standards:

- The ODBC driver complies with version 3.5.1 of the ODBC standard.
- Vertica's JDBC driver is a type 4 driver that complies with the JDBC 3.0 standard. It is compiled using JDK version 1.5, and is compatible with client applications compiled using JDK versions 1.5 and 1.6.
- ADO.NET drivers conform to .NET framework 3.0 specifications.

The drivers do not support some of the optional features in the standards. See ODBC Feature Support and JDBC Feature Support and Using ADO.NET for details.
Client Driver and Server Version Compatibility

Usually, each version of the Vertica server is compatible with the previous version of the client drivers. This compatibility lets you upgrade your Vertica server without having to immediately upgrade your client software. However, some new features of the new server version may not be available through the old drivers.

Note: Vertica Release 9.0.x and later adds backwards compatibility for the ODBC client driver. The 9.0.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see Data Source Name (DSN) Connection Properties.

The following tables summarize the compatibility of each recent version of the client drivers with the Vertica server versions.

The following table indicates that, in general, all clients are forward compatible.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Clients</td>
<td>6.1.x</td>
<td>6.1.x, 7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>7.0.x</td>
<td>7.0.x, 7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>7.1.x</td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>7.2.x</td>
<td>7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>8.0.x</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>8.1.x</td>
<td>8.1.x, 9.0.x</td>
</tr>
<tr>
<td></td>
<td>9.0.x</td>
<td>9.0.x</td>
</tr>
</tbody>
</table>

The following table lists FIPS 140-2 compatible clients.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIPS-enabled ODBC</td>
<td>8.0.x</td>
<td>8.0.x, 8.1.x</td>
</tr>
<tr>
<td>FIPS-enabled ODBC</td>
<td>8.1.x</td>
<td>8.0.x, 8.1.x</td>
</tr>
</tbody>
</table>
The following table indicates that the ODBC client is backward compatible.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIPS-enabled ODBC</td>
<td>9.0.x</td>
<td>8.0.x, 8.1.x, 9.0.x</td>
</tr>
<tr>
<td>FIPS-enabled JDBC</td>
<td>8.1.x</td>
<td>8.1.x</td>
</tr>
<tr>
<td>FIPS-enabled JDBC</td>
<td>9.0.x</td>
<td>8.1.x, 9.0.x</td>
</tr>
</tbody>
</table>

The following table indicates that the JDBC and ADO.NET clients are backward compatible.

<table>
<thead>
<tr>
<th>Client</th>
<th>Client Driver Version</th>
<th>Compatible Server Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC (backwards compatibility)</td>
<td>8.1.x</td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x</td>
</tr>
<tr>
<td>ODBC (backwards compatibility)</td>
<td>9.0.x</td>
<td>7.1.x, 7.2.x, 8.0.x, 8.1.x, 9.0.x</td>
</tr>
</tbody>
</table>

Vertica ODBC/JDBC Client Installers

The ODBC/JDBC client drivers are a separate installation from the ADO.NET drivers. (ADO.NET support is not available in Community Edition.) As noted in the compatibility table, the 6.x ODBC/JDBC client drivers do not support access to a non Vertica 6.x database and above. For example, you cannot use the new 6.x ODBC/JDBC client drivers to access a Vertica 5.x database. If you plan on having a mixed Vertica environment supporting both 5.x and 6.x Vertica database, consider keeping the 5.x drivers installed.

ODBC/JDBC Multiple Version Installations

The following ODBC/JDBC drivers are supported on a single machine:
- 4.x and 5.x ODBC/JDBC drivers can be installed on the same machine.
- 4.x and 6.x ODBC/JDBC drivers can be installed on the same machine.

It is not possible to have both 5.x and 6.x ODBC drivers on a single machine. If you install the 6.x version, it automatically overlays the existing 5.x installation, and any DSN defined against a 5.x Vertica database is not supported.

**Vertica ADO.NET Client Installers**

Prior to version 6.x, ADO.Net drivers must be uninstalled prior to installing a later version of the driver. The 6.x ADO.Net drivers require the Vertica database to be 6.0.0 or above. The ADO.NET 6.x driver only supports access to a Vertica 6.x server. The ADO.NET 4.x plug-in does not work with a Vertica 6.x server. If you plan on also using the ODBC bridge and you need to access both Vertica 5.x and 6.x databases, consider keeping the 5.x versions of the ODBC/JDBC drivers for the reasons stated previously.
Client Drivers

You must install the Vertica client drivers to access Vertica from your client application. The drivers create and maintain connections to the database and provide APIs that your applications use to access your data. The client drivers support connections using JDBC, ODBC, and ADO.NET.

Client Driver Standards

The client drivers support the following standards:

- ODBC drivers conform to ODBC 3.5.1 specifications.
- JDBC drivers conform to JDK 5 specifications.
- ADO.NET drivers conform to .NET framework 3.0 specifications.

Installing the Client Drivers

How you install client drivers depends on the client's operating system:

- For Linux and UNIX clients, you must first install a Linux driver manager. After you have installed the driver manager, there are two different ways to install the client drivers:
  
  - On Red Hat Enterprise Linux 5, 64-bit and SUSE Linux Enterprise Server 10/11 64-bit, you can use the Vertica client RPM package to install the ODBC and JDBC drivers as well as the vsql client.
  
  - On other Linux platforms and UNIX-like platforms you can download the ODBC and JDBC drivers and install them individually.

Note: The ODBC and JDBC client drivers are installed by the server .rpm files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX). To use JDBC, you need to add the JDBC
client driver to the Java CLASSPATH (see Modifying the Java CLASSPATH). (The JDBC client driver is not available on FIPS-compliant systems.)

- On Mac OS X clients, download the ODBC client driver . pkg file. The driver is compatible with both 32-bit and 64-bit applications.

- On Windows clients, download the 32-bit or 64-bit client installer. The installer provides the ODBC client driver, the ADO.NET client driver, the OLE DB client driver, the vsql client, the Microsoft Connectivity Pack, and the Visual Studio plug-in.

- There is a cross-platform JDBC client driver . jar file available for installation on all platforms.

The remainder of this section explain the requirements for the Vertica client drivers, and the procedure for downloading, installing, and configuring them.

Driver Prerequisites

The following topics describe the system requirements for the client drivers. You need to ensure that your client system meets these requirements before installing and using the client drivers.

ODBC Prerequisites

There are several requirements your client systems must meet before you can install the Vertica ODBC drivers.

Operating System

The Vertica ODBC driver requires a supported platform. The list of currently-supported platforms can be found at Vertica 9.0.x Client Drivers.

ODBC Driver Manager

The Vertica ODBC driver requires that the client system have a supported driver manager. See Installing Driver Managers Linux and Other UNIX-like Platforms for details.
UTF-8, UTF-16 and UTF-32 Support

The Vertica ODBC driver is a universal driver that supports UTF-8, UTF-16, and UTF-32 encoding. The default setting depends on the client platform. For details, see Required ODBC Driver Configuration Settings for Linux and UNIX.

When using the driver with the DataDirect Connect driver manager, DataDirect Connect adapts to the ODBC driver's text encoding settings. You should configure the ODBC driver to use the encoding method that your application requires. This allows strings to be passed between the driver and the application without intermediate conversion.

See Also

- Client Drivers
- Programming ODBC Client Applications
- Creating an ODBC Data Source Name (DSN)

ADO.NET Prerequisites

The Vertica driver for ADO.NET requires the following software and hardware components:

Operating System

The Vertica ADO.NET driver requires a supported Windows operating system. The list of supported platforms can be found in the Supported Platforms document at http://my.vertica.com/docs.

Memory

Vertica suggests a minimum of 512MB of RAM.

.NET Framework

The requirements for the .NET framework for ADO.NET in Vertica can be found in the Supported Platforms document at http://my.vertica.com/docs.
See Also

- Programming ADO.NET Applications

Python Prerequisites

Python is a free, agile, object-oriented, cross-platform programming language designed to emphasize rapid development and code readability. Python has been released under several different open source licenses.

Vertica's ODBC driver is tested with multiple versions of Python. See Perl and Python Requirements for details.

Python Driver

Vertica requires the Vertica Python Client or the pyodbc driver module. See your system's Python documentation for installation and configuration information.

Supported Operating Systems

The Vertica ODBC driver requires one of the operating systems listed in ODBC Prerequisites. For usage and examples, see Programming Python Client Applications.

Perl Prerequisites

Perl is a free, stable, open source, cross-platform programming language licensed under its Artistic License, or the GNU General Public License (GPL).

Your Perl scripts access Vertica through its ODBC driver, using the Perl Database Interface (DBI) module with the ODBC Database Driver (DBD::ODBC). The Vertica ODBC driver is known to be compatible with these versions of Perl:

- 5.8
- 5.10

Later Perl versions may also work.
Perl Drivers

The following Perl driver modules have been tested with the Vertica ODBC driver:

- The DBI driver module, version 1.609
- The DBD::ODBC driver module, version 1.22

Other versions may also work.

Supported Client Systems

The Vertica ODBC driver requires one of the operating systems and driver managers listed in ODBC Prerequisites.

PHP Prerequisites

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP is licensed under the PHP License, an open-source BSD-style license certified by the Open Source Initiative.

PHP Modules

The following PHP modules are required:

- php
- php-odbc
- php-pdo
- UnixODBC (if you are using the Unix ODBC driver)
- libioddbc (if you are using the iODBC driver)

Supported Client Systems

The Vertica ODBC driver requires one of the operating systems and driver managers listed in ODBC Prerequisites.
Upgrading the Client Drivers

The Vertica client drivers are usually updated for each new release of the Vertica server. The client driver installation packages include the version number of the corresponding Vertica server release. Usually, the drivers are forward-compatible with the next release, so your client applications are still able to connect using the older drivers after you upgrade to the next version of Vertica Analytics Platform server. See Client Driver and Server Version Compatibility for details on which client driver versions work with each version of Vertica server.

Note: Vertica Release 9.0.x and later adds backwards compatibility for the ODBC client driver. The 9.0.x ODBC client driver is backwards compatible to Vertica server version 7.1. For full compatibility with the previous server version, specify the Protocol property in your connection string. For more information about the Protocol property, see Data Source Name (DSN) Connection Properties.

You should upgrade your clients as soon as possible after upgrading your server, to take advantage of new features and to maintain maximum compatibility with the server. To upgrade your drivers, follow the same procedure you used to install them in the first place. The new installation will overwrite the old. See the specific instructions for installing the drivers on your client platform for any special instructions regarding upgrades.

Note: Installing new ODBC drivers does not alter existing DSN settings. You may need to change the driver settings in either the DSN or in the odbcinst.ini file, if your client system uses one. See Creating an ODBC Data Source Name for details.

Setting a Client Connection Label

You can set a client connection label when you connect to a Vertica database. You can also set or return the client connection label using the SET_CLIENT_LABEL and GET_CLIENT_LABEL functions.

Set the client connection label:

```sql
=> SELECT SET_CLIENT_LABEL('py_data_load_application');
SET_CLIENT_LABEL
----------------------
client_label set to py_data_load_application
(1 row)
```

Return the current client connection label:

...
The JDBC Client has a method to set and return the client connection label: `getClientInfo()` and `setClientInfo()`. You can use these methods with the SQL Functions `GET_CLIENT_LABEL` and `SET_CLIENT_LABEL`.

When you use these two methods, make sure you pass the string value `APPLICATIONNAME` to both the setter and getter methods.

Use `setClientInfo()` to create a client label, and use `getClientInfo()` to return the client label:

```java
import java.sql.*;
import java.util.Properties;

public class ClientLabelJDBC {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "");
        myProp.put("loginTimeout", "35");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://docc05.verticacorp.com:5433/doccdb", myProp);
            System.out.println("Connected!");
            conn.setClientInfo("APPLICATIONNAME", "JDBC Client - Data Load");
            System.out.println("New Conn label: " + conn.getClientInfo("APPLICATIONNAME"));
            conn.close();
        } catch (SQLTransientConnectionException connException) {
            System.out.println("Network connection issue:");
            System.out.println(connException.getMessage());
            return;
        } catch (SQLException e) {
            System.out.println("Could not log into database:");
            System.out.println(e.getMessage());
            System.out.println("Check the login credentials and try again.");
            return;
        }
    }
}
```
When you run this method, it prints the following result to the standard output:

```
Connected!
New Conn Label: JDBC Client - Data Load
```

### Additional Parameter Settings

The following parameters can be set for the Vertica client drivers.

### Logging Settings

These parameters control how messages between the client and server are logged. None of these settings are required. If they are not set, then the client library does not log any messages. They apply to both ADO.NET and ODBC.

- **LogLevel**—The severity of messages that are logged between the client and the server. The valid values are:
  - 0—No logging
  - 1—Fatal errors
  - 2—Errors
  - 3—Warnings
  - 4—Info
  - 5—Debug
  - 6—Trace (all messages)

The value you specify for this setting sets the minimum severity for a message to be logged. For example, setting LogLevel to 3 means that the client driver logs all warnings, errors, and fatal errors.

**Note:** On a Windows client, you have the option of directing ODBC or OLE DB log entries to Event Tracing for Windows (ETW). Once set, ODBC log entries appear in the Windows Event Viewer. See Register the ODBC Driver as a Windows Event Log Provider, and
Enable the Logs for ODBC, or Register the OLE DB Driver as a Windows Event Log Provider, and Enable the Logs for OLE DB.

- **LogPath**—The absolute path of a directory to store log files. For example: /var/log/verticaodbc

- **LogNamespace**—Limits logging to messages generated by certain objects in the client driver.

**Note:** These settings are also available for the Vertica JDBC driver through connection properties. See Connection Properties for details.

### ODBC-specific Settings

The following settings are used only by the Vertica ODBC client driver.

- **DriverManagerEncoding**—The UTF encoding standard that the driver manager uses. This setting needs to match the encoding the driver manager expects. The available values for this setting are:
  - UTF-8
  - UTF-16 (usually used by unixODBC)
  - UTF-32 (usually used by iODBC)

See the documentation for your driver manager to find the correct value for this setting.

**Note:** While both UTF-16 and UTF-8 are valid settings for DataDirect, Vertica recommends that you set the DataDirect driver manager encoding to UTF-16.

If you do not set this parameter, the ODBC driver defaults to the value shown in the following table. If your driver manager uses a different encoding, you must set this value for the ODBC driver to be able to work.

<table>
<thead>
<tr>
<th>Client Platform</th>
<th>Default Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>HP-UX 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Client Platform</td>
<td>Default Encoding</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Linux 32-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Linux 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Linux Itanium 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>OS X</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Solaris 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit</td>
<td>UTF-32</td>
</tr>
<tr>
<td>Windows 32-bit</td>
<td>UTF-16</td>
</tr>
<tr>
<td>Windows 64-bit</td>
<td>UTF-16</td>
</tr>
</tbody>
</table>

- **ErrorMessagePath**—The absolute path to the parent directory that contains the Vertica client driver's localized error message files. These files are usually stored in the same directory as the Vertica ODBC driver files.

  Note: This setting is required. If you do not set it, then any error the ODBC driver encounters will result in an error message about a missing ODBCMessages.xml file.

- **ODBCInstLib**—The absolute path to the file containing the ODBC installer library (ODBCInst). This setting is required if the directory containing this library is not set in the LD_LIBRARY_PATH or LIB_PATH environment variables. The library files for the major driver manager are:
  - UnixODBC: libodbcinst.so
  - iODBC: libiodbcinst.so (libiodbcinst.2.dylib on OS X)
  - DataDirect: libodbcinst.so

  Note: On AIX platforms, you need give the path to the library archive, followed by the name of the library enclosed in parenthesis. For example: `ODBCInstLib=/usr/lib64/libodbcinst.a(libodbcinst.so.1)`

**ADO.NET-specific Settings**

This setting applies only to the ADO.NET client driver:
C#PreloadLogging—Tells the Vertica ADO.NET driver to begin logging as soon as possible, before the driver has fully loaded itself. Normally, logging only starts after the driver has fully loaded. Valid values for this setting are:

- 0—Do not start logging before the driver has loaded.
- 1—Start logging as soon as possible.

Using Legacy Drivers

The Vertica server supports connections from the previous version of the client drivers. For example, the Vertica version 5.1 server works with the 4.1 client drivers, since they were the drivers distributed with the previous version of the server. This backwards compatibility lets you upgrade your Vertica database first, then later upgrade your clients.

If you have not yet updated your code to work with the new version of the Vertica client drivers, you can continue to use the older drivers until you do. If you need to install your client application on a new client system, you can download and install the older drivers. See myVertica portal to download the installers; find installation documentation at http://my.vertica.com/docs.

For detailed information on which the compatibility of different versions of the Vertica server and Vertica client, see Client Driver and Server Version Compatibility.

Note: The support for a previous version of the drivers is usually eliminated in the next release of Vertica. For example, the Vertica version 5.1 server does not support the version 4.0 drivers. You should update your client application to work with the new client drivers as soon as possible.

Modifying the Java CLASSPATH

The CLASSPATH environment variable contains the list of directories where the Java run time looks for library class files. For your Java client code to access Vertica, you need to add the directory where the Vertica JDBC .jar file is located.

Note: In your CLASSPATH, use the symbolic link vertica-jdbc-x.x.x.jar (where x.x.x is a version number) that points to the JDBC library .jar file, rather than the .jar file itself. Using the symbolic link ensures that any updates to the JDBC library .jar file (which will use a different filename) will not invalidate your CLASSPATH setting, since the symbolic link's filename will remain the same. You just need to update the symbolic link to point at the new .jar file.
Linux, Solaris, AIX, HP-UX, and OS X

If you are using the Bash shell, use the `export` command to define the CLASSPATH variable:

```bash
# export CLASSPATH=/opt/vertica/java/lib/vertica-jdbc-x.x.x.jar
```

If environment variable CLASSPATH is already defined, use the following command to prevent it from being overwritten:

```bash
# export CLASSPATH=$CLASSPATH:/opt/vertica/java/lib/vertica-jdbc-x.x.x.jar
```

If you are using a shell other than Bash, consult its documentation to learn how to set environment variables.

You need to either set the CLASSPATH environment variable for every login session, or insert the command to set the variable into one of your startup scripts (such as `~/.profile` or `/etc/profile`).

Windows

Provide the class paths to the `.jar`, `.zip`, or `.class` files.

```bash
C:> SET CLASSPATH=classpath1;classpath2...
```

For example:

```bash
C:> SET CLASSPATH=C:\java\MyClasses\vertica-jdbc-x.x.x.jar
```

As with the Linux/UNIX settings, this setting only lasts for the current session. To set the CLASSPATH permanently, set an environment variable:

1. On the Windows Control Panel, click System.
2. Click Advanced or Advanced Systems Settings.
3. Click Environment Variables.
5. In the Variable name box, type CLASSPATH.
6. In the Variable value box, type the path to the Vertica JDBC `.jar` file on your system (for example, `C:\Program Files (x86)\Vertica\JDBC\vertica-jdbc-x.x.x.jar`).
Specifying the Library Directory in the Java Command

There is an alternative way to tell the Java run time where to find the Vertica JDBC driver other than changing the CLASSPATH environment variable: explicitly add the directory containing the .jar file to the java command line using either the -cp or -classpath argument. For example, on Linux, start your client application using:

```
# java -classpath /opt/vertica/java/lib/vertica-jdbc-x.x.x.jar myapplication.class
```

Your Java IDE may also let you add directories to your CLASSPATH, or let you import the Vertica JDBC driver into your project. See your IDE documentation for details.

Installing the Client Drivers on Linux and UNIX-Like Platforms

This topic details how to install the client drivers on Linux and Unix-like platforms.

Installing Driver Managers Linux and Other UNIX-like Platforms

If your client platform does not already have an ODBC driver manager, you need to install one before you can use the Vertica ODBC client driver. The driver manager provides an interface between your client operating system and the ODBC drivers. See Vertica 9.0.x Client Drivers for a list of driver managers that are supported on each of the client platforms.

Driver managers can be downloaded from your operating system specific repository and from the links below.

Vertica does not provide instructions for installing and configuring these third party binaries. For download and configuration information, see the respective websites for the driver managers for installation and configuration information:

- iODBC: [http://www.iodbc.org](http://www.iodbc.org)

Installing the Client RPM on Red Hat and SUSE

For Red Hat Enterprise Linux and SUSE Linux Enterprise Server, you can download and install a client driver RPM that installs both the ODBC and JDBC driver libraries and the vsql client.
Important: Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsq1 and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, Installing the FIPS Client Driver for ODBC and vsq1.

To install the client driver RPM package:

1. Open a Web browser and log in to the myVertica portal.

2. Click Downloads, and choose Client Drivers.

3. Download the client RPM file that matches your client platform's architecture.

   Note: The 64-bit client driver RPM installs both the 64-bit and 32-bit ODBC driver libraries on non-FIPS compliant systems.

4. If you did not directly download the RPM on the client system, transfer the file to the client.

5. Log in to the client system as root.

6. Install the RPM package you downloaded:

   # rpm -Uvh package_name.rpm

   Note: You receive one or more conflict error messages if there are existing Vertica client driver files on your system. This can happen if you are trying to install the client driver package on a system that has the server package installed, since the server package also includes the client drivers. In this case, you don't need to install the client drivers, and can instead use the drivers installed by the server package. If the conflict arises from an old driver installation or from a server installation for an older version, you can use the rpm command's --force switch to force it to overwrite the existing files with the files in the client driver package.

Once you have installed the client package, you need to create a DSN (see Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX) and set some additional configuration parameters (see Required ODBC Driver Configuration Settings for Linux and UNIX) to use ODBC. To use JDBC, you need to modify your class path (see Modifying the Java CLASSPATH) before you can use JDBC.

You may also want to add the vsq1 client to your PATH environment variable so that you do not need to enter its full path to run it. You add it to your path by adding the following to the .profile file in your home directory or the global /etc/profile file.
export PATH=$PATH:/opt/vertica/bin

Installing JDBC Driver on Linux, Solaris, AIX, and HP-UX

**Note:** The ODBC and JDBC client drivers are installed by the server .rpm files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX). To use JDBC, you need to add the JDBC client driver to the Java CLASSPATH (see Modifying the Java CLASSPATH). (The JDBC client driver is not available on FIPS-compliant systems.)

**Note:** For additional details about supported platforms, see Supported Platforms.

The JDBC driver is available for download from myVertica portal. There is a single .jar file that works on all platforms and architectures. To download and install the file:

1. Open a Web browser and log in to myVertica portal.
2. Click the Download tab and locate and download the JDBC driver.
3. You need to copy the .jar file you downloaded to a directory in your Java CLASSPATH on every client system with which you want to access Vertica. You can either:
   - Copy the .jar file to its own directory (such as /opt/vertica/java/lib) and then add that directory to your CLASSPATH (recommended). See Modifying the Java CLASSPATH for details.
   - Copy the .jar file to directory that is already in your CLASSPATH (for example, a directory where you have placed other .jar files on which your application depends).

**Note:** In the directory where you copied the .jar file, you should create a symbolic link named vertica_jdk_5.jar to the .jar file. You can reference this symbolic link anywhere you need to use the name of the JDBC library without having to worry any future upgrade invalidating the file name. This symbolic link is automatically created on server installs. On clients, you need to create and manually maintain this symbolic link yourself if you installed the driver manually. The Installing the Client RPM on Red Hat and SUSE create this link when they install the JDBC library.
Installing ODBC Drivers on Linux, Solaris, AIX, and HP-UX

**Note:** For additional details about supported platforms, see [Supported Platforms](#).

Read [Driver Prerequisites](#) before you proceed.

For Red Hat Enterprise Linux and SUSE Linux Enterprise Server, you can download and install a client RPM that installs both the ODBC and JDBC driver as well as the vsql client. See [Installing the Client RPM on Red Hat and SUSE](#).

**Note:** The ODBC and JDBC client drivers are installed by the server .rpm files. If you have installed Vertica Analytics Platform on your Linux system for development or testing purposes, you do not need to download and install the client drivers on it—you just need to configure the drivers. To use ODBC, you need to create a DSN (see [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX](#)). To use JDBC, you need to add the JDBC client driver to the Java CLASSPATH (see [Modifying the Java CLASSPATH](#)). (The JDBC client driver is not available on FIPS-compliant systems.)

The ODBC driver installation packages are broken down by client platform on the [myVertica portal](#). The package's filename is named based on its operating system and architecture (for example, `vertica_9.0..xx_odbc_x86_64_linux.tar.gz`)

**Important:** Vertica provides the FIPS-compliant client driver only as an rpm for 64-bit clients. You can install this rpm only on FIPS-enabled machines. The FIPS client includes vsql and ODBC drivers. If you are installing the FIPS-specific client, refer to the section, [Installing the FIPS Client Driver for ODBC and vsql](#).

**Installation Procedure**

1. Open a Web browser and log in to [myVertica portal](#).

2. Click the Download tab and locate and download the driver package that corresponds to your client system.

3. If you did not directly download to the client system, transfer the downloaded file to it.

4. Log in to the client system as root.

5. If the directory `/opt/vertica/` does not exist, create it:
# mkdir -p /opt/vertica/

6. Copy the downloaded file to the /opt/vertica/ directory. For example:

   # cp vertica_9.0_xx_odbc_x86_64_linux.tar.gz /opt/vertica/

7. Change to the /opt/vertica/ directory:

   # cd /opt/vertica/

8. Uncompress the file you downloaded. For example:

   $ tar vzxf vertica_9.0_xx_odbc_x86_64_linux.tar.gz

   Two folders are created: one for the include file, and one for the library files. The path of the library file depends on the processor architecture: lib for 32-bit libraries, and lib64 for 64-bit libraries. So, a 64-bit driver client download creates the directories:

   - /opt/vertica/include, which contains the header file
   - /opt/vertica/lib64, which contains the library file

Post Driver Installation Configuration

You must configure the ODBC driver before you can use it. There are two required configuration files:

- The odbc.ini configuration file defines the Data Source Names (DSNs) that tell the ODBC how to access your Vertica databases. See Creating an ODBC Data Source Name for instructions to create this file.

- The vertica.ini configuration file defines some Vertica-specific settings required by the drivers. See Required ODBC Driver Configuration Settings for Linux and UNIX for instructions to create this file.

Note: If you are upgrading your ODBC driver, you must either update your DSNs to point to the newly-installed driver or create new DSNs. If your odbc.ini file references drivers defined in an odbcinst.ini file, you just need to update the odbcinst.ini file. See Creating an ODBC Data Source Name (DSN) for details.
Required ODBC Driver Configuration Settings for Linux and UNIX

In addition to DSN settings, Vertica provides additional ODBC client driver configuration parameters. These settings control the following:

- The text encoding used by the driver manager (for example, UTF-8 or UTF-16).
- The location of the directory containing the Vertica ODBC driver’s error message files.
- Whether and how the ODBC driver logs messages.

On Linux and UNIX platforms, you must provide these additional settings manually so that the ODBC driver can function properly. To do so, edit the vertica.ini file to supply the necessary additional configuration settings. You specify where the ODBC driver can find the vertica.ini file using an environment variable named VERTICAINI. See Required Settings.

Setting ODBC Driver Settings on Linux and UNIX-Like Platforms

Driver settings specific to Vertica are stored in a text file named vertica.ini (although you may choose a different file name). On Linux and UNIX platforms, you must edit the vertica.ini file to supply additional configuration settings before the ODBC driver can function properly. You tell the Vertica ODBC driver where to find the vertica.ini file using an environment variable named VERTICAINI.
Required Settings

On Linux and UNIX platforms, you must configure two settings in order for the ODBC driver to work correctly:

- `ErrorMessagePath`
- `ODBCInstLib` (unless the driver manager's installation library is in a directory listed in the `LD_LIBRARY_PATH` or `LIB_PATH` environment variables).

If your driver manager does not use UTF-8 encoding, you need to set `DriverManagerEncoding` to the proper encoding.
Create a vertica.ini File

There is no standard location for the vertica.ini file—you can store the file anywhere that it is convenient for you on your client system. One possible location is in the /etc directory if you have multiple users on your client system that need to access it. You can also have a vertica.ini file in each user's home directory so users can alter their own settings. Wherever you store it, be sure users have read access to the file.

The format of the vertica.ini file is similar to the odbc.ini file, with a section followed by parameter definitions. Unlike the odbc.ini file, vertica.ini contains a single section named Driver:

    [Driver]

Following the section definition, you add setting definitions, one per line. A setting definition consists of the setting name, followed by an equal sign (=), followed by the value. The value does not need quotes. For example, to set the ODBCInstLib setting, you add a line like this:

    ODBCInstLib=/usr/lib64/libodbcinst.so

See Additional Parameter Settings for a list of the additional settings.
Set the VERTICAINI Environment Variable

You must set an environment variable named VERTICAINI to the absolute path of the vertica.ini file. The Vertica ODBC driver uses this variable to find the settings.

Where you set this variable depends on whether users on your client system need to have separate vertica.ini files. If you want to have a single, system-wide vertica.ini file, you can add a command to set VERTICAINI in /etc/profile or some other system-wide environment file. For example:

```
export VERTICAINI=/etc/vertica.ini
```

If users need individual vertica.ini files, set VERTICAINI in their ~/.profile or similar configuration file. For example:

```
export VERTICAINI=~/.vertica.ini
```
Example vertica.ini File

The following example vertica.ini file configures the ODBC driver to:

- use the 64-bit UNIXODBC driver manager.
- get its error messages from the standard Vertica 64-bit ODBC driver installation directory.
- log all warnings and more severe messages to log files stored in the temporary directory.

```
[Driver]
DriverManagerEncoding=UTF-16
ODBCInstLib=/usr/lib64/libodbcinst.so
ErrorMessagePath=/opt/vertica
LogLevel=4
LogPath=/tmp
```

Installing the FIPS Client Drivers

This topic details how to install the FIPS client drivers for JDBC and ODBC.

Installing the FIPS Client Driver for ODBC and vsql

Vertica offers a FIPS client for FIPS-compatible systems. A FIPS-compatible system is FIPS-enabled and includes the OpenSSL libraries.

The FIPS client supports ODBC and vsql and is offered in 64-bit only.

Prerequisites

Verify that your host system is running Red Hat Enterprise Linux 6.6.

The FIPS client installer checks your host system for the value of the sysctl parameter, crypto.fips_enabled. You must set this parameter to 1 (enabled). If your host is not enabled, the client does not install.

For other prerequisites, related specifically to ODBC, see ODBC Prerequisites.

Installing the FIPS Client

To install the FIPS client driver package:
1. Download the FIPS client package from the myVertica portal.

2. Log in to the client system as root.

3. Install the RPM package you downloaded:

   ```
   # rpm -Uvh package_name.rpm
   ```

For ODBC, once you have installed the client package, you need to create a DSN and set some additional configuration parameters. For more information, see:

- Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX
- Required ODBC Driver Configuration Settings for Linux and UNIX

You may also want to add the vsq1 client to your PATH environment variable so that you do not need to enter its full path to run it. To do so, add the following to the .profile file in your home directory or the global /etc/profile file:

   ```
   export PATH=$PATH:/opt/vertica/bin
   ```

**Client Searches for OpenSSL Libraries**

When you launch the client application to connect to the server, the client searches for and loads the OpenSSL libraries libcrypto.so.10 and libssl.so.10 (version 1.0.1e):

- The client first checks to see if LD_LIBRARY_PATH is set.

- If the LD_LIBRARY_PATH location does not include the libraries, it checks RunPath, either /opt/vertica/lib or within the ODBC or vsq1 directory structure (/..lib).

   **Important:** The LD_LIBRARY_PATH, if set, directs the search path for the OpenSSL libraries. Be aware that the client loads the libraries from any set or preset LD_LIBRARY_PATH location.

The following figure depicts the search for the OpenSSL libraries.
Installing the FIPS Client Driver for JDBC

Vertica offers a JDBC client driver that is compliant with the Federal Information Processing Standard (FIPS). Use this JDBC client driver to access systems that are FIPS-compatible. For more information on FIPS in Vertica, see Federal Information Processing Standard.

Implementing FIPS on a JDBC client requires a third-party JRE extension called BouncyCastle, a collection of APIs used for cryptography. Use BouncyCastle APIs with JDK 1.7 and 1.8, and Red Hat 6.6.
Important: When using the JDBC FIPS-compliant client, expect some time lag for the client to connect efficiently and securely. If necessary, increase your system's entropy to ensure a fast and secure connection.

You need to add the FIPS BouncyCastle jar as the JVM JSSE provider, as follows:

1. Download the BouncyCastle FIPS jar file bc-fips-1.0.0.jar from the BouncyCastle download page.

2. Add bc-fips-1.0.0.jar as a JRE library extension:

   `<path to jre>/lib/ext/bc-fips-1.0.0.jar`

3. Add BouncyCastle as an SSL security provider in `<path to jre>/lib/security/java.security`:

   ```
   security.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
   security.provider.2=com.sun.net.ssl.internal.ssl.Provider BCFIPS
   security.provider.3=sun.security.provider.Sun
   ```

4. Use the following JVM java -D system property command arguments to set the KeyStore and TrustStore files to BCFIPS:

   ```
   export JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.keyStoreProvider=BCFIPS
   export JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStoreProvider=BCFIPS
   ```

   For information on setting the SSL Keystore and Truststore, see Configuring SSL for JDBC Clients.

5. Set the default type for the KeyStore implementation to BCFKS in `<path to jre>/lib/security/java.security`:

   ```
   keystore type=BCFKS
   ssl.keystore.type=BCFKS
   ```

Note: If you are using FIPS with BouncyCastle, you must create all client keys and certificates with the BCFKS store type, including the Vertica–Kafka key/certs.
6. On the command line, run the following command from <path to jre>/lib/ext to create the keystore and truststore. Make sure you use the BCFKS type:

```
$ <java bin path> keytool -keystore vertica.kafka.keystore.bcfks
   -storetype BCFKS
   -providername BCFIPS
   -providerclass
   org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
   -providerorg.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
   -providerpath bc-fips-1.0.0.jar
   -alias CARoot
   -import -file (server.crt.der file path)
```

7. Enter the keystore password when prompted. The following message appears:

"Certificate was added to the keystore"

8. Run the java program with SSL DB:
   a. Copy the vertica.kafka.keystore.bcfks keyStore from <path to jre>/lib/ext/ to the java program folder.
   b. Convert the Vertica server certificate to a form that java understands:
      i. openssl x509  
         -in server.crt  
         -out server.crt.der  
         -outform der
      ii. <java bin path>/keytool  
          -keystore verticastore  
          -keystorepwd 'password'  
          -keystore 'vertica.kafka.keystore.bcfks'  
          -truststore <path to verticastore>/verticastore  
          -truststorepwd 'password'
   c. Download the latest vertica jdbc driver from the my.vertica.com download page.

9. After creation of verticastore, keyStore, and download jar, execute the following command to run java with debugging to test the implementation:

```
$ java -Djavax.net.debug=ssl
-Djavax.net.ssl.keyStore='vertica.kafka.keystore.bcfks'
-Djavax.net.ssl.keyStorePassword='password'
-Djavax.net.ssl.trustStore='<path to verticastore>/verticastore'
-Djavax.net.ssl.trustStorePassword='password'
-cp .:vertica-jdbc-8.1.0-0.jar FIPSTest
```
Installing the Client Drivers and Tools on Windows

This section details how to install the client drivers and tools on Windows.

For connectivity through a JDBC connection, see Installing the JDBC Client Driver for Windows.
For all other client drivers and tools, see The Vertica Client Drivers and Tools for Windows.

Installing the JDBC Client Driver for Windows

To install the Vertica JDBC driver on your Windows client system, you must first download the cross-platform JDBC driver .jar file to your system. Then, choose the method that the Windows Java installation will use to find it.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsq1, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 8.1.x to 9.0.x would not require that you uninstall before upgrade.

Download the JDBC Driver for Windows

1. On your Windows client system, open a browser, and log in to the myVertica portal.

2. Install the Vertica JDBC driver for Windows:
   a. Navigate to the Downloads tab, and scroll to the Client Software section.
   b. Click the download link for the JDBC Driver for Windows installer.

3. Accept the license agreement, and wait for the download to complete.

Choose How Java Locates the JDBC Driver Library

For your Java client application to use the Vertica JDBC driver, the Java interpreter must be able to find the driver's library file. Choose one of these methods to tell the Java interpreter where to look for the library:
Copy the JDBC .jar file you downloaded to the system-wide Java Extensions folder (C:\Windows\Sun\Java).

Add the directory containing the JDBC .jar file to the CLASSPATH environment variable (see Modifying the Java CLASSPATH).

Specify the directory containing the JDBC .jar using the -cp argument in the Java command line you use to start your Java command line.

The Vertica Client Drivers and Tools for Windows

You can obtain the Vertica Client Drivers and Tools for Windows installer through the myVertica portal. You can run the installer on either a 32-bit or 64-bit system, either as a regular Windows installer or silently.

Components

The Vertica Client Drivers and Tools for Windows installs the following components on your system:

- The ODBC Client Driver for Windows
- The OLE DB Client Driver for Windows
- The vsql Client for Windows
- The ADO.NET Driver for Windows
- The Microsoft Connectivity Pack for Windows
- The Visual Studio Plug-in for Windows

Read Fully Update Your System before you proceed.

System Prerequisites

The Vertica Client Drivers and Tools for Windows released with Vertica Release 7.2. or later has basic system prerequisite requirements. The pack also requires that specific Microsoft components be installed for full integration.

For a list of all prerequisites, see Vertica 9.0.x Client Drivers in the Supported Platforms document.
Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsq1, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 8.1.x to 9.0.x would not require that you uninstall before upgrade.
**Fully Update Your System**

Before you install the Vertica driver package, verify that your system is fully up to date with all Windows updates and patches. See the documentation for your version of Windows for instructions on how to run Windows update. The Vertica client libraries and vsql executable install updated Windows libraries that depend on Windows service packs. Be sure to resolve any issues that block the installation of Windows updates.

If your system is not fully up-to-date, you may receive error messages about missing libraries such as `api-ms-win-crt-runtime-l1-1-0.dll` when starting vsql.
.NET Framework

The .NET framework is not bundled into the Vertica Client Drivers and Tools for Windows. However, during installation, a web installer launches if Microsoft .NET 3.5 SP1 is not detected on your system. You then have the opportunity to download the framework. Also, if your operating system version includes .NET 3.5 SP1, but it is not turned on, the installer turns on the feature.

If you have Visual Studio 2010 or 2012 installed, your system already includes Microsoft .NET Framework 4.0 or 4.5, respectively. You also need Microsoft .NET 3.5 SP1 to use the Vertica Client Drivers and Tools for Windows integration features.

Use the following links to download the appropriate version of .NET framework directly from Microsoft:

- For .NET Framework 3.5 SP1:
  

- For .NET Framework 4.0:


- For .NET Framework 4.5:

Microsoft Visual Studio


Important: You must have Visual Studio and the matching SDK installed to use the Visual Studio plug-in.

After installing the plug-in, you can use it to access your Vertica database from within Visual Studio. If you do not have the SDK installed, download the SDK specific to your version of Visual Studio.

Note: For Visual Studio 2015, you do not need to download the SDK separately as it is included as an installation option with the Visual Studio installation. For more information, refer to the Microsoft documentation.

- For the Microsoft Visual Studio 2008 SDK:

- For the Microsoft Visual Studio 2008 SP1 SDK:

- For the Microsoft Visual Studio 2010 SDK:

- For the Microsoft Visual Studio 2010 SP1 SDK:

- For the Microsoft Visual Studio 2012 SDK:

- For the Microsoft Visual Studio 2013 SDK:

If the Microsoft Visual Studio SDK is missing when you begin the installation, a dialog box opens to tell you so. You can choose to ignore this dialog box.
### Configuring BIDS or SSDT-BI Integration

The Vertica Client Drivers and Tools for Windows installer provides BIDS (Visual Studio 2008) or SSDT-BI (Visual Studio 2010, 2012, 2013, or 2015) integration. To use BIDS or SSDT-BI, follow this process:

1. Install the BIDS or SSDT-BI development tool add-on for Visual Studio.

2. Verify that SQL Server is installed on the same or a different machine.

3. Verify that the SQL Server Shared Features for BIDS or SSDT-BI have been activated.

You can then develop packages using BIDS or SSDT-BI, creating your projects using SQL Server’s SSIS, SSAS, SSRS features. To use these features, you must connect to Vertica through the Vertica ADO.NET driver (for SSIS and SSRS) or the OLE DB driver (for SSAS).

For more information, see [Microsoft Components](#).
Microsoft SQL Server

Use SQL Server 2012, 2014 or 2016. The Vertica Client Drivers and Tools for Windows installer enables support for the following:

- SQL Server 2012, 2014, and 2016:
  - SQL Server Integration Services (SSIS)
  - SQL Server Reporting Services (SSRS)
  - SQL Server Analysis Services (SSAS)


Note: For SQL Server 2012, you can use either SQL Server 2012 or SQL Server 2012 SP1.

To use the enhanced Vertica .NET support, you must first install SQL Server. Then, you can install the Client Drivers and Tools for Windows. The following components must be installed on the SQL server:

<table>
<thead>
<tr>
<th>For...</th>
<th>Install...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSAS</td>
<td>The Analysis Services Instance Feature.</td>
</tr>
<tr>
<td>SSRS</td>
<td>The Reporting Services Instance Feature.</td>
</tr>
<tr>
<td>SSIS (Data Type Mappings)</td>
<td>The SQL Server Integration Services Shared Feature.</td>
</tr>
</tbody>
</table>
Download the Client Drivers and Tools for Windows

1. On your Windows client system, open a browser, and log in to the myVertica portal.

2. Install the Vertica Client Drivers and Tools for Windows:
   a. Navigate to the Downloads tab, and scroll to the Client Software section. Click the link for the Vertica Client Drivers download page.
   b. Click the download link for the Vertica Client Drivers and Tools installer.

3. Accept the license agreement, and wait for the download to complete.

Install or Upgrade the Client Drivers and Tools for Windows

As the Windows Administrator, double-click the installer to start the installation. Follow the prompts as the wizard guides you through each step of the process.

By default, the installer installs the client drivers and tools in C:\Program Files\Vertica Systems\. You have the option of changing this location during installation.

**Note:** The install wizard allows you to choose the components you want to install. Under your Windows Control Panel, the installer adds a program for each installed component. Among the programs the installer adds is a program named, Vertica Client Installer. You can right-click on the installer to modify, repair, or uninstall the Vertica client.

**Important:** The Vertica Microsoft Connectivity Pack is included as part of the Vertica Client Drivers and Tools for Windows. If you plan to use the Microsoft Connectivity Pack to access Microsoft Business Intelligence tools, reboot your system after installation to ensure integration.
Upgrading the Client Drivers and Tools for Windows

You do not need to uninstall the Client Drivers and Tools for Windows before upgrading. The installation program upgrades the existing drivers and tools in place.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsql, ADO.NET, OLEDB, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 8.1.x to 9.0.x would not require that you uninstall before upgrade.

Silently Install or Upgrade the Client Drivers and Tools for Windows

1. As a Windows Administrator, open a command-line session, and change directory to the folder that contains the installer.

2. Run the command:

   ```
   VerticaSetup.exe -q -install InstallFolder="C:\Program Files\Vertica Systems"
   ```

The client drivers and tools are silently installed in C:\Program Files\Vertica Systems\.
Upgrading the Client Drivers and Tools for Windows

You do not need to uninstall the Client Drivers and Tools for Windows before upgrading. The installation program upgrades the existing drivers and tools in place.

Important: If you are using Vertica client drivers for Windows that were released before Vertica 7.2.3, you must first uninstall the older drivers. This is true for the separate JDBC client driver download (the .jar file). This is also true for the client drivers included in the package, Client Drivers and Tools for Windows (an .exe file), which includes ODBC, vsq1, ADO.NET, OLED, the Visual Studio plugin, and SQL Server integration components. Once you have installed client versions 7.2.3, you do not need to uninstall for upgrades that follow. For example, upgrading from 8.1.x to 9.0.x would not require that you uninstall before upgrade.

Post-Installation Steps for ODBC Driver and vsq1 Client

After you install the Vertica Client Drivers and Tools for Windows, there are additional steps you must take for the ODBC driver and vsq1 client to function correctly.

- **ODBC Driver** — After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see Creating an ODBC DSN for Windows Clients.

- **vsq1 Client** — The vsq1 client does not have a shortcut. Before you can start using vsq1, you must add the vsq1 executable to the Windows PATH environment variable. The method for altering the PATH environment variable depends on the version of the Microsoft Windows operating system you are running. To start vsq1 and show the help list, open a command window, and type vsq1 -? at the command prompt. See Using vsq1 for Windows Users for important details about using vsq1 in a Windows console.

Uninstalling, Modifying, or Repairing the Client Drivers and Tools

To uninstall, modify, or repair the client drivers and tools, run the Client Drivers and Tools for Windows installer.

The installer provides three options:
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify</td>
<td>Remove installed client drivers and tools or install missing client drivers and tools.</td>
</tr>
<tr>
<td>Repair</td>
<td>Reinstall already-installed client drivers and tools.</td>
</tr>
<tr>
<td>Uninstall</td>
<td>Uninstall all of the client drivers and tools.</td>
</tr>
</tbody>
</table>
Silently Uninstall the Client Drivers and Tools

1. As a Windows Administrator, open a command-line session, and change directory to the folder that contains the installer.

2. Run the command:

   \texttt{VerticaSetup.exe -q -uninstall}

The client drivers and tools are silently uninstalled.

Components of the Client Drivers and Tools on Windows

The following sections describe the components in the Client Drivers and Tools for Windows in more detail:

- The ODBC Client Driver for Windows
- The vsql Client for Windows
- The Microsoft Connectivity Pack for Windows
- The OLE DB Client Driver for Windows
- The ADO.NET Driver for Windows
- The Visual Studio Plug-in for Windows

The ODBC Client Driver for Windows

The Vertica ODBC driver for Windows is installed as part of the Client Drivers and Tools for Windows.

After Installing the ODBC Driver

After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see \textit{Creating an ODBC DSN for Windows Clients}. 
ODBC Driver Settings on Windows

ODBC driver settings are automatically configured using the Vertica Client Drivers and Tools installer on Windows. The values for the settings are stored in the Windows registry under the path HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\ODBC\Driver. It is not necessary to configure additional ODBC driver settings on Windows platforms beyond what is automatically configured by the installer. You can, however, set the ODBC driver settings using the Windows ODBC Data Source Configuration window.

See Additional Parameter Settings for a list of additional settings for the ODBC client driver. See Register the ODBC Driver as a Windows Event Log Provider, and Enable the Logs for information on how to send ODBC log entries to Event Tracing for Windows (ETW).

Diverting ODBC Log Entries to ETW

On Windows clients, you can direct Vertica to send ODBC log entries to Event Tracing for Windows (ETW). Once set, ODBC log entries appear in the Windows Event Viewer. To use ETW:

- Register the driver as a Windows Event Log provider, and enable the logs.
- Activate ETW by adding a string value to your Windows Registry.
- Understand how Vertica compresses log levels for the Windows Event Viewer.
- Know where to find the logs within Event Viewer.
- Understand the meaning of the Event IDs in your log entries.
Register the ODBC Driver as a Windows Event Log Provider, and Enable the Logs

To use ETW logging, you must register the ODBC driver as a Windows Event Log provider. You can choose to register either the 32-bit or 64-bit driver. Once you have registered the driver, you must enable the logs.

**Important:** If you do not both register the driver and enable the logs, output is directed to stdout.

1. Open a command prompt window as Administrator, or launch the command prompt with the Run as Administrator option.

   **Important:** You must have administrator privileges to successfully complete the next step.

2. Run the command `wevtutil im` to register either the 32-bit or 64-bit version of the driver.

   a. For the 64-bit ODBC driver, run:
   ```
   wevtutil im "c:\Program Files\Vertica Systems\ODBC64\lib\VerticaODBC64.man"
   /resourceFilePath:"c:\Program Files\Vertica Systems\ODBC64\lib\vertica_9.0_odbc_3.5.dll"
   /messageFilePath:"c:\Program Files\Vertica Systems\ODBC64\lib\vertica_9.0_odbc_3.5.dll"
   
   b. For the 32-bit ODBC driver, run:
   ```
   ```
   wevtutil im "c:\Program Files (x86)\Vertica Systems\ODBC32\lib\VerticaODBC32.man"
   /resourceFilePath:"c:\Program Files (x86)\Vertica Systems\ODBC32\lib\vertica_9.0_odbc_3.5.dll"
   /messageFilePath:"c:\Program Files (x86)\Vertica Systems\ODBC32\lib\vertica_9.0_odbc_3.5.dll"
   ```

3. Run the command `wevtutil sl` to enable the logs.

   a. For 64-bit ODBC driver logs, run:
   ```
   wevtutil sl VerticaODBC64/e:true
   ```
b. For the 32-bit ODBC driver logs, run:

```
wevtutil sl VerticaODBC32/e:true
```

**Note:** Should you want to later disable the logs, you can use the same `wevtutil sl` command, substituting `/e:false` in place of `/e:true` when you issue the statement. Alternatively, you can enable or disable logs within the Windows Event Viewer itself.
Add the String Value LogType

By default, Vertica does not send ODBC log entries to ETW. To activate ETW, add the string LogType to your Windows registry, and set its value to ETW.

1. Start the registry editor by typing regedit.exe in the Windows Run command box.
2. Navigate to the correct location in the registry.
   
   | HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\ODBC\Driver |
   
3. Right-click in the right pane of the Registry Editor window. Select New and then select String Value.
4. Change the name of the string value from New Value #1 to LogType.
5. Double-click the new LogType entry. When prompted for a new value, enter ETW.
6. Exit the registry editor.

ETW is off by default. When ETW is activated, you can subsequently turn it off by clearing the value ETW from the LogType string.
# Event Viewer Log Levels

The LogLevel parameter setting is described in the section, Additional Parameter Settings. The parameter allows you to specify a LogLevel of 0 through 6. Be aware that Vertica compresses the log levels for the Windows Event Viewer. The six levels are compressed to four in Event Viewer.

<table>
<thead>
<tr>
<th>Vertica LogLevel Setting</th>
<th>Vertica LogLevel Description</th>
<th>Entries are sent to Event Viewer as log level...</th>
<th>Event Viewer Displays...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(No logging)</td>
<td>0</td>
<td>(No logging)</td>
</tr>
<tr>
<td>1</td>
<td>Fatal Errors</td>
<td>1</td>
<td>Critical</td>
</tr>
<tr>
<td>2</td>
<td>Errors</td>
<td>2</td>
<td>Error</td>
</tr>
<tr>
<td>3</td>
<td>Warnings</td>
<td>3</td>
<td>Warning</td>
</tr>
<tr>
<td>4</td>
<td>Info</td>
<td>4</td>
<td>Information</td>
</tr>
<tr>
<td>5</td>
<td>Debug</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Trace (all messages)</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

- A LogLevel setting of 5 sends fatal errors, errors, warnings, info and debug log level entries to Event Viewer as Level 4 (Information).

- A LogLevel setting of 6 sends fatal errors, errors, warnings, debug and trace log level entries to Event Viewer as Level 4.
Where to Find Logs in Event Viewer

1. Launch the Windows Event Viewer.

2. From Event Viewer (Local), expand Applications and Services Logs.

3. Expand the folder that contains the log you want to review (for example, VerticaODBC64).

4. Select the Vertica ODBC log under the folder. Entries appear in the right pane.
Event Log Entry: Event ID

Once you have chosen an ODBC log in Event Viewer, note the value in the Event ID field.

Each Event Log entry includes one of four Event IDs. An Event ID of 0 is informational (debug, info, and trace events), 1 is an error, 2 is a fatal event, and 3 is a warning.

The vsql Client for Windows

The Vertica vsql client for Windows is installed as part of the Client Drivers and Tools for Windows.

There is no shortcut for the vsql client. Before you can start using vsql, you must add the vsql executable to the Windows PATH environment variable. The method for altering the PATH environment variable depends on the version of the Microsoft Windows operating system you are running. After you have made the change to your PATH environment variable, start a command window and type vsql -? at the command prompt to start vsql and show the help list.

See Using vsql for Windows Users for important details about using vsql in a Windows console.

Using vsql for Windows Users
Font

The default raster font does not work well with the ANSI code page. Set the console font to "Lucida Console."
Console Encoding

vsqI is built as a "console application." The Windows console windows use a different encoding than the rest of the system, so take care when you use 8-bit characters within vsqI. If vsqI detects a problematic console code page, it warns you at startup.

To change the console code page, set the code page by entering cmd.exe /c chcp 1252.

Note: 1252 is a code page that is appropriate for European languages. Replace it with your preferred locale code page.
Running Under Cygwin

Verify that your cygwin.bat file does not include the "tty" flag. If the "tty" flag is included in your cygwin.bat file, then banners and prompts are not displayed in vsql.

To verify, enter:

```bash
set CYGWIN=binmode tty ntsec
```

To remove the "tty" flag, enter:

```bash
set CYGWIN=binmode ntsec
```

Additionally, when running under Cygwin, vsql uses Cygwin shell conventions as opposed to Windows console conventions.
Tab Completion

Tab completion is a function of the shell, not vsq1. Because of this, tab completion does not work the same way in Windows vsq1 as it does on Linux versions of vsq1.

On Windows, instead of using tab-completion, press F7 to pop-up a history window of commands. You can also press F8 after typing a few letters of a command to cycle through commands in the history buffer which begin with the same letters.

The Microsoft Connectivity Pack for Windows

The Vertica Microsoft Connectivity Pack for Windows provides a configuration file for you to access Microsoft Business Intelligence tools. The Connectivity Pack is installed as part of the Client Drivers and Tools for Windows.

To learn about which Microsoft components are configured with the Microsoft Connectivity Pack, see Microsoft Components.

Microsoft Components

This section describes the Microsoft Business Intelligence components you can use with Microsoft Visual Studio and Microsoft SQL Server. After configuration, you can use these Microsoft components to develop business solutions using your Vertica server.

Important: The Vertica Microsoft Connectivity Pack is included as part of the Vertica Client Drivers and Tools for Windows. If you plan to use the Microsoft Connectivity Pack to access Microsoft Business Intelligence tools, reboot your system after installation to ensure integration.
Microsoft Component Configuration

The Vertica ADO.NET driver, the Visual Studio plug-in, and the OLE DB driver allow you to integrate your Vertica server with an environment that includes Microsoft components previously installed on your system. Additional tools are also available for integration with Microsoft SQL Server.

The available drivers provide integration with the following Microsoft components:

- **Business Intelligence Development Studio (BIDS) for Visual Studio 2008** for use with SQL Server 2012. BIDS is a client-based application used to develop business intelligence solutions based on the Microsoft Visual Studio development environment. It includes additional project types specific to SQL Server Business Intelligence. As a developer, you can use BIDS to develop business solutions.


- **SQL Server Analysis Services (SSAS)** for SQL Server 2012, 2014, and 2016. Use SSAS for OLAP and data mining, while using Vertica as the source for cube creation.

- **SQL Server Integration Services (SSIS)** for SQL Server 2012, 2014, and 2016. SSIS provides SQL Server Type Mappings to map data types between Vertica and SQL Server. Use SSIS for data migration, data integration and workflow, and ETL.

The following figure displays the relationship between Microsoft components and Vertica dependencies.
BIDS and SSDT-BI

Business Intelligence Development Studio (BIDS) is available in Microsoft Visual Studio 2008 with additional project types that are specific to SQL Server business intelligence. BIDS is the primary environment that you use to develop business solutions that include Analysis Services, Integration Services, and Reporting Services projects.


Both BIDS and SSDT-BI are client-based applications that include additional project types specific to SQL Server Business Intelligence.

You can use the Visual Studio Shell Integration plug-in to browse a database from within the Visual Studio Server Explorer. This capability allows you to work outside of BIDS or SSDT-BI development to perform tasks, such as listing tables or inserting data. When you use Visual Studio in BIDS or SSDT-BI mode, you can develop business solutions using the data in your Vertica database. For example, you can create cubes or open tables.

Microsoft does not support the following configurations:

- You cannot use Microsoft Visual Studio 2008 with BIDS development to create a SQL Server 2012 Business Intelligence solution.

- You cannot use Microsoft Visual Studio 2010/2012/2013/2015 with SSDT-BI development to create a SQL Server 2008 Business Intelligence solution.
SQL Server Analysis Services (SSAS) Support

BIDS or SSDT-BI includes the Analysis Services project for developing online analytical processing (OLAP) for business intelligence applications. This project type includes templates for:

- Cubes
- Dimensions
- Data sources
- Data source views

It also provides the tools for working with these objects.

Note: OpenText recommends that you use the Vertica OLE DB driver when connecting to the Vertica server from SSAS due to improved performance.

You can find the OLE DB connection properties in OLE DB Connection Properties.
SQL Server Integration Services (SSIS) Support

BIDS or SSDT-BI includes the Integration Services project for developing ETL solutions. This project type includes templates for:

- Packages
- Data sources
- Data source views

It also provides the tools for working with these objects.

You can find support for using Vertica as a data source and target from both SSIS and the import/export wizard. You must install mapping files specific to Vertica on the Integration Server and BIDS or SSDT-BI workstation to enable this capability. The Vertica Client Drivers and Tools for Windows installs these mapping files as the "SQL Server Type Mappings" component(s) in both 32-bit and 64-bit versions.

Note: Always use the Vertica ADO.NET driver when connecting to the Vertica server from SSIS.
SQL Server Reporting Services (SSRS) Support

BIDS or SSDT-BI includes Report projects for developing reporting solutions.

You can use Vertica as a data source for Reporting Services. The installer implements various configuration file modifications to enable this capability on both the BIDS or SSDT-BI workstation and the Reporting Services server.

Compatibility Issues and Limitations

This section lists compatibility issues and limitations for integrating the Microsoft Connectivity Pack with Microsoft Visual Studio and Microsoft SQL Server.
BIDS and SSDT-BI Limitations

BIDS and SSDT-BI are 32-bit development environments for Analysis Services, Integration Services, and Reporting Services projects. They are not designed to run on the Itanium 64-bit architecture and thus are not installed on Itanium servers.
SSAS Limitations

- The SSAS Tabular Model is not supported.

- If, after installing the Vertica OLE DB driver, an SSAS cube build fails, restart the SSAS service.
SSIS Data Type Limitations

The following sections cover data type limitations when using SQL Server Integration Services (SSIS).
Time Data Transfer

When transferring time data, SSIS uses the TimeSpan data type that supports precision greater than six digits. The Vertica ADO.NET driver translates TimeSpan as an Interval data type that supports up to six digits. The Interval type is not converted to the TimeSpan type during transfer. As a result, if the time value has a precision of more than six digits, the data is truncated, not rounded.

For information on ADO.NET data types, refer to ADO.NET Data Types.
DATE and DATETIME Precision

To function without errors, DATE and DATETIME have a range from 0001-01-01 00:00:00.0000000 to 9999-12-31 23:59:59.999999.

In SSIS, the DATETIME type (DT_TIMESTAMP) supports only a scale up to three decimal places for seconds. Any decimal places after that are automatically discarded. You can perform derived column transformations only on DATETIME values between January 1, 1753 through December 31, 9999.
Numeric Precision

The maximum and minimum decimal allowed is:

- Max: +79,228,162,514,265,93,543,950,335
- Min: -79,228,162,514,264,337,593,543,950,335

For example, if the scale is 16, the range of values is:

+/- 7,922,816,251,426,433,7593,543950335

The valid scale range is any number that is smaller than 29 and greater than 38. Using a scale between 29 and 38 does not generate an error.

Unsupported Floating Point Values

SQL Server does not support NaN, Infinity, or –Infinity values. These values are supported when you use SSIS to transfer between Vertica instances, but they are not supported with a SQL Server Destination.
String Conversion

The CHAR and VARCHAR data types used in SSIS are DT_WSTR, with a maximum length of 4000 characters.

In SSIS, Vertica strings are converted to Unicode strings in SSIS to handle multi-lingual data. You can convert these strings to ASCII using a Data Conversion Task.
Scale

Whenever you use a scale greater than 38, SSIS replaces it with a value of 4.
Interval Conversion

SSIS does not support interval types. It converts them to TIME and strips off the day component. Any package that has interval types greater than a day returns incorrect results.
Data Mapping Issues with SQL Server Import and Export Wizard

When you create an Integrated Services package (SSIS) using the SQL Server Import and Export Wizard, certain data types do not automatically map correctly. A mapping issue can occur when you use the wizard with:

- SQL Server Native OLE DB Provider for SQL Server 2008 or 2012
- SQL Server Native Client 10.0/11.0 Provider for SQL Server 2010/2012

To avoid this issue, manually change the type mappings with BIDS or SSDT-BI.
Data Transfer Failures

When using an Integrated Services package (SSIS) with the SQL Server OLE DB Provider for SQL Server 2008 or 2012, certain data type transfers can fail when transferring from Vertica to SQL Server. To avoid this issue, use either BIDS or SSDT-BI when transferring data.
Batch Insert of VARBINARY/LONG VARBINARY Data Types

Sometimes, one row of a batch insert of VARBINARY or LONG VARBINARY data types exceeds the data type limit:

- VARBINARY: 65 KB
- LONG VARBINARY: 32 MB

In such cases, all rows are rejected, rather than just the one row whose length exceeds the type limit. The batch insert fails with the message "row(s) are rejected".

To avoid this issue, use a predicate to filter out rows from the source that do not fit into the receiving database.
Boolean Queries in SQL Server Query Designer

When issuing a Boolean query in SQL Server Query Designer, you must enclose Boolean column values in quotes. Otherwise, you receive a SQL execution error (for example, someboolean = 'true').
SSRS Limitations

Data Connection Wizard Workaround

The SSRS Report Wizard provides a data connection wizard. After you select the wizard and enter all the connection information, the OK button is disabled. You cannot save your work and continue. The workaround is to not use the wizard and to use the following panel instead:

![Report Wizard - Query Designer](image)

Vertica uses the Report Wizard’s Generic Query Designer. Other data sources use a Graphical Query Designer that supports building queries visually. The Graphical Query Designer is a part of a package called Visual Data Tools (VDT). The Graphical Query Designer works only with Generic OLE DB providers and the built-in providers. You cannot use it with the Vertica Data Provider.

Report Builder

Report Builder is a web-based report design tool. It does not support creating reports using custom data extensions, so you cannot use it with Vertica. When you create a report using Report Builder, existing Vertica data sources appear in the list of available data sources. However, choosing a Vertica data source causes an error.

Schema Name Not Automatically Provided when Mapping Vertica Destination

Currently, when you map a Vertica destination, the schema name is not automatically provided. You must enter it manually or pick it from the drop-down menu as follows:
The OLE DB Client Driver for Windows

The Vertica OLE DB driver for Windows is installed as part of the Client Drivers and Tools for Windows. The values for the OLE DB driver's settings are stored in the Windows registry under the path HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\OLEDB\Driver.

For information on how the OLE DB driver integrates with Microsoft components previously installed on your system, see Microsoft Component Configuration.

OLE DB Connection Properties

Use the Connection Manager to set the OLE DB connection string properties, which define your connection. You access the Connection Manager from within Visual Studio.

These connection parameters appear on the Connection page.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Action</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
<td>Select the native OLE DB provider for the connection.</td>
<td>None</td>
</tr>
<tr>
<td>OLE DB Provider</td>
<td>Indicates Vertica OLE DB Provider.</td>
<td>None</td>
</tr>
<tr>
<td>Server or file name</td>
<td>Enter the server or file name.</td>
<td>None</td>
</tr>
<tr>
<td>Parameters</td>
<td>Action</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>Location</td>
<td>Not supported.</td>
<td>Disabled.</td>
</tr>
<tr>
<td>Use Windows NT Integrated Security</td>
<td>Not supported.</td>
<td>Disabled.</td>
</tr>
<tr>
<td>Use a specific user name and password</td>
<td>Enter a user name and password.</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>Connect with No Password:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select the Blank password check box.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Save and Encrypt Password:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select Allow saving password.</td>
<td></td>
</tr>
<tr>
<td>Initial Catalog</td>
<td>The name of the database running on the server.</td>
<td>None</td>
</tr>
</tbody>
</table>

The All page from the Connection Manager dialog box includes all possible connection string properties for the provider.

The table that follows lists the connection parameters for the All page.

For OLE DB properties information specific to Microsoft, see the Microsoft documentation [OLE DB Properties](#).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Action</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Properties</td>
<td>Not supported.</td>
<td>Leave blank. Do not set this field.</td>
</tr>
<tr>
<td>Locale Identifier</td>
<td>Indicates the Locale ID.</td>
<td>0</td>
</tr>
<tr>
<td>Mode</td>
<td>Specifies access permissions.</td>
<td>0</td>
</tr>
<tr>
<td>Connect Timeout</td>
<td>Not supported. The value can be set, but has no effect.</td>
<td>0</td>
</tr>
<tr>
<td>General Timeout</td>
<td>Not supported. The value can be set, but has no effect.</td>
<td>0</td>
</tr>
<tr>
<td>File Name</td>
<td>Not supported. The value can be set, but has no effect.</td>
<td>Blank</td>
</tr>
<tr>
<td>Parameters</td>
<td>Action</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>set, but has no effect.</td>
<td></td>
</tr>
<tr>
<td>OLE DB Services</td>
<td>Specifies which OLE DB services to enable or disable.</td>
<td>Default</td>
</tr>
<tr>
<td>Password</td>
<td>Specifies the password for the User ID.</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>For no password, insert an empty string.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(If no password specified, login succeeds only if the user has not set a password.)</td>
<td></td>
</tr>
<tr>
<td>Persist Security Info</td>
<td>A security measure. When False, security sensitive-information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state.</td>
<td>True</td>
</tr>
<tr>
<td>User ID</td>
<td>The database username.</td>
<td>None</td>
</tr>
<tr>
<td>Data Source</td>
<td>The host name or IP address of any active node in a Vertica cluster.</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>You can provide an IPv4 address, IPv6 address, or host name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.</td>
<td></td>
</tr>
<tr>
<td>Initial Catalog</td>
<td>The name of the database running on the server.</td>
<td>None</td>
</tr>
<tr>
<td>Provider</td>
<td>The name of the OLE DB Provider to use when connecting to the Data Source.</td>
<td>VerticaOLEDB.1</td>
</tr>
<tr>
<td>Parameters</td>
<td>Action</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>BackupServerNode</td>
<td>A designated host name or IP address to use if the ServerName host is unavailable. Enter as a string. Connection attempts continue until successful or until the list of server nodes is exhausted. Valid values: Comma-separated list of servers optionally followed by a colon and port number. For example: server1:5033,server2:5034</td>
<td>None</td>
</tr>
<tr>
<td>ConnectionLoadBalance</td>
<td>A Boolean value that determines whether the connection can be redirected to a host in the database other than the ServerNode. This parameter affects the connection only if load balancing is set to a value other than NONE. When the node differs from the node that the client is connected to, the client disconnects and reconnects to the targeted node. See About Native Connection Load Balancing in the Administration Guide.</td>
<td>False</td>
</tr>
</tbody>
</table>
| ConnSettings             | SQL commands that the driver should execute immediately after connecting to the server. Use to configure the connection, such as setting a schema search path. Reserved symbol: ';' To set multiple parameters in this field use '%3B' for ','.

**Spaces:** Use '+' | None            |
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Action</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvertSquareBracketIdentifiers</td>
<td>Controls whether square-bracket query identifiers are converted to a double quote identifier for compatibility when making queries to a Vertica database.</td>
<td>False</td>
</tr>
<tr>
<td>DirectBatchInsert</td>
<td>Controls where data inserted through the connection is stored.</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Values:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- True — Data inserted directly into ROS containers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- False — Stores data using AUTO mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</td>
<td></td>
</tr>
<tr>
<td>KerberosHostName</td>
<td>Provides the instance or host name portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM</td>
<td>None</td>
</tr>
<tr>
<td>KerberosServiceName</td>
<td>Provides the service name portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM</td>
<td>None</td>
</tr>
<tr>
<td>Label</td>
<td>Sets a label for the connection on the server. This value appears in the session_id column of the V_MONITOR.SESSIONS system table.</td>
<td>None</td>
</tr>
<tr>
<td>LogLevel</td>
<td>Specifies the amount of information included in the log.</td>
<td>None</td>
</tr>
<tr>
<td>Parameters</td>
<td>Action</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Leave this field blank or set to 0 unless otherwise instructed by Vertica Customer Support.</td>
<td></td>
</tr>
<tr>
<td>LogPath</td>
<td>The path for the log file.</td>
<td>None</td>
</tr>
<tr>
<td>Port</td>
<td>The port number on which Vertica listens for OLE DB connections.</td>
<td>None (If not set, uses port 5433.)</td>
</tr>
<tr>
<td>PreferredAddressFamily</td>
<td>The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>* ipv4—Connect to the server using IPv4.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* ipv6—Connect to the server using IPv6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>* none—Use the IP address provided by the DNS server.</td>
<td></td>
</tr>
<tr>
<td>SSLCertFile</td>
<td>The absolute path of the client's public certificate file. This file can reside anywhere on the system.</td>
<td>None</td>
</tr>
<tr>
<td>SSLKeyFile</td>
<td>The absolute path to the client's private key file. This file can reside anywhere on the system.</td>
<td>None</td>
</tr>
<tr>
<td>SSLMode</td>
<td>Controls whether the connection to the database uses SSL encryption. Valid values follow. Valid Values:</td>
<td>Prefer</td>
</tr>
<tr>
<td></td>
<td>* require—Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails.</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefer—Prefer that the server use SSL. If the server does not offer an encrypted channel, the client requests one. The first attempt is made with SSL. If that attempt fails, the second attempt is over a clear channel.</td>
<td></td>
</tr>
<tr>
<td>allow—Makes a connection to the server whether or not the server uses SSL. The first attempt is made over a clear channel. If that attempt fails, a second attempt is over SSL.</td>
<td></td>
</tr>
<tr>
<td>disable—Never connects to the server using SSL. Typically, you use this setting for troubleshooting.</td>
<td></td>
</tr>
</tbody>
</table>

**Diverting OLE DB Log Entries to ETW**

On Windows clients, you can direct Vertica to send OLE DB log entries to Event Tracing for Windows (ETW). Once set, OLE DB log entries appear in the Windows Event Viewer. To use ETW:

- Register the driver as a Windows Event Log provider, and enable the logs.
- Activate ETW by adding a string value to your Windows Registry.
- Understand how Vertica compresses log levels for the Windows Event Viewer.
- Know where to find the logs within Event Viewer.
- Understand the meaning of the Event IDs in your log entries.
Register the OLE DB Driver as a Windows Event Log Provider, and Enable the Logs

To use ETW logging, you must register the OLE DB driver as a Windows Event Log provider. You can choose to register either the 32-bit or 64-bit driver. Once you have registered the driver, you must enable the logs.

Important: If you do not both register the driver and enable the logs, output is directed to stdout.

1. Open a command prompt window as Administrator, or launch the command prompt with the Run as Administrator option.

   Important: You must have administrator privileges to successfully complete the next step.

2. Run the command `wevtutil im` to register either the 32-bit or 64-bit version of the driver.

   a. For the 64-bit OLE DB driver, run:

      ```
      wevtutil im "c:\Program Files\Vertica Systems\OLEDB64\lib\VerticaOLEDB64.man"
      /resourceFilePath:"c:\Program Files\Vertica Systems\OLEDB64\lib\vertica_8.1_oledb.dll"
      /messageFilePath:"c:\Program Files\Vertica Systems\OLEDB64\lib\vertica_8.1_oledb.dll"
      ```

   b. For the 32-bit OLE DB driver, run:

      ```
      wevtutil im "c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\VerticaOLEDB32.man"
      /resourceFilePath:"c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\vertica_8.1_oledb.dll"
      /messageFilePath:"c:\Program Files (x86)\Vertica Systems\OLEDB32\lib\vertica_8.1_oledb.dll"
      ```

3. Run the command `wevtutil sl` to enable the logs.

   a. For 64-bit OLE DB driver logs, run:

      ```
      wevtutil sl VerticaOLEDB64/e:true
      ```
b. For the 32-bit ODBC driver logs, run:

```
wevtutil sl VerticaOLEDB32/e:true
```

Note: Should you want to later disable the logs, you can use the same `wevtutil sl` command, substituting `/e:false` in place of `/e:true` when you issue the statement. Alternatively, you can enable or disable logs within the Windows Event Viewer itself.
Add the String Value LogType

By default, Vertica does not send OLE DB log entries to ETW. To activate ETW, add the string LogType to your Windows registry, and set its value to ETW.

1. Start the registry editor by typing regedit.exe in the Windows Run command box.

2. Navigate to the correct location in the registry.

   HKEY_LOCAL_MACHINE\SOFTWARE\Vertica\OLEDB\Driver

3. Right-click in the right pane of the Registry Editor window. Select New and then select String Value.

4. Change the name of the string value from New Value #1 to LogType.

5. Double-click the new LogType entry. When prompted for a new value, enter ETW.

6. Exit the registry editor.

ETW is off by default. When ETW is activated, you can subsequently turn it off by clearing the value ETW from the LogType string.
Event Viewer Log Levels

The LogLevel parameter setting is described in the section, Additional Parameter Settings. The parameter allows you to specify a LogLevel of 0 through 6. Be aware that Vertica compresses the log levels for the Windows Event Viewer. The six levels are compressed to four in Event Viewer.

<table>
<thead>
<tr>
<th>Vertica LogLevel Setting</th>
<th>Vertica LogLevel Description</th>
<th>Entries are sent to Event Viewer as log level...</th>
<th>Event Viewer Displays...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(No logging)</td>
<td>0</td>
<td>(No logging)</td>
</tr>
<tr>
<td>1</td>
<td>Fatal Errors</td>
<td>1</td>
<td>Critical</td>
</tr>
<tr>
<td>2</td>
<td>Errors</td>
<td>2</td>
<td>Error</td>
</tr>
<tr>
<td>3</td>
<td>Warnings</td>
<td>3</td>
<td>Warning</td>
</tr>
<tr>
<td>4</td>
<td>Info</td>
<td>4</td>
<td>Information</td>
</tr>
<tr>
<td>5</td>
<td>Debug</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Trace (all messages)</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

- A LogLevel setting of 5 sends fatal errors, errors, warnings, info and debug log level entries to Event Viewer as Level 4 (Information).
- A LogLevel setting of 6 sends fatal errors, errors, warnings, debug and trace log level entries to Event Viewer as Level 4.
Where to Find Logs in Event Viewer

1. Launch the Windows Event Viewer.

2. From Event Viewer (Local), expand Applications and Services Logs.

3. Expand the folder that contains the log you want to review (for example, VerticaOLEDB64).

4. Select the Vertica OLE DB log under the folder. Entries appear in the right pane.
Event Log Entry: Event ID

Once you have chosen an OLE DB log in Event Viewer, note the value in the Event ID field.
Each Event Log entry includes one of four Event IDs. An Event ID of 0 is informational (debug, info, and trace events), 1 is an error, 2 is a fatal event, and 3 is a warning.

The ADO.NET Driver for Windows

The Vertica ADO.NET driver for Windows is installed as part of the Client Drivers and Tools for Windows.

The ADO.NET driver is installed in the ADO.NET folder of the installation folder. The driver is also installed into the Windows Global Assembly Cache (GAC).

For information on how the ADO.NET driver integrates with Microsoft components previously installed on your system, see Microsoft Components.

The Visual Studio Plug-in for Windows

The Visual Studio plug-in for Windows is installed as part of the Client Drivers and Tools for Windows.

For information on how the Visual Studio plug-in integrates with Microsoft components previously installed on your system, see Microsoft Components.

Visual Studio Limitations

Visual Studio 2012 May Require Update 3

You may need to install update 3 to Visual Studio 2012 if:

- You launch Server Explorer to view and work with your Vertica server, but the Vertica data source is not visible.
- You create a SSAS cube, connect to Vertica, and find either an empty list of tables or tables not functioning correctly.

This issue does not occur for other versions of Visual Studio supported by Vertica.

Results Viewer Limited to 655 Columns
The Visual Studio results viewer cannot execute a query that includes more than 655 columns. If a table includes more than 655 columns, select specific columns (up to 655 total) rather than selecting all columns.

Manually Refresh Settings for Visual Studio

If, after installing the Visual Studio plug-in, you do not see Vertica listed as a data provider, manually refresh.

To do so, run devenv.exe/setup, which you can find in the Visual Studio installation folder.

SQL Pane Issues

- **ALTER TABLE or CREATE TABLE**

  You use Visual Studio 2008, 2010, 2012, 2013, or 2015 and issue the ALTER TABLE or CREATE TABLE statement in the SQL pane. However, a message displays telling you that the statement is not supported. To resolve the error, click Continue, and the query executes.

- **Queries with Semicolons**

  You use Visual Studio 2008, 2010, 2012, 2013, or 2015 and execute a SQL query in the SQL pane. If you include a semicolon (;) with your query, the query executes, but the result returned cannot be edited. To avoid this issue, enter the same query in the SQL pane without the semicolon.

- **Quoting Boolean Values**

  You use Visual Studio 2008, 2010, 2012, 2013, or 2015 to connect to the Vertica database and execute a SQL query in the SQL pane. When attempting to insert a value into a Boolean column without putting quotes around the value, subsequent execution of the SQL statement returns an error. To work around this issue, include quotes.

Uninstalling **Client Drivers and Tools for Windows** Error

There is a scenario where an uninstall of the Client Drivers and Tools for Windows package fails with a message that the .NET framework is required. What follows is the scenario that causes this issue.

1. You Install the Client Drivers and Tools for Windows.

2. You then install Visual Studio 2010 or 2012, which includes installation of the .NET framework 4.0 or 4.5.

3. You uninstall the .NET framework using the Windows Control Panel.
4. You then attempt to uninstall the Client Drivers and Tools for Windows. The uninstall fails with the message that .NET framework is required.

Perform the following to correct this issue:

1. Reinstall the .NET framework 4.0 or 4.5 manually, using the Windows Control Panel.
2. Uninstall the Client Drivers and Tools for Windows.

**Installing the Client Drivers on Mac OS X**

This section details how to install the client drivers on Mac OS X.

**Installing the JDBC Driver on Mac OS X**

To install the Vertica JDBC driver on your Mac OS X client system, download the cross-platform JDBC driver .jar file to your system and ensure OS X's Java installation can find it.

**Downloading the JDBC Driver**

To download the Vertica JDBC driver on Mac OS X:

1. On your Mac client system, open a browser and log into the myVertica portal.
2. Navigate to the Downloads page, scroll to the Client Software download section, and click the download link for the JDBC driver.
3. Accept the license agreement and wait for the download to complete.

**Ensuring Java Can Find the JDBC Driver**

In order for your Java client application to use the Vertica JDBC driver, the Java interpreter needs to be able to find its library file. Choose one of these methods to tell the Java interpreter where to look for the library:

- Copy the JDBC .jar file you downloaded to either the system-wide Java Extensions folder (/Library/Java/Extensions) or your user Java Extensions folder (/Users/username/Library/Java/Extensions).
Add the directory containing the JDBC .jar file to the CLASSPATH environment variable (see Modifying the Java CLASSPATH).

Specify the directory containing the JDBC .jar using the -cp argument in the Java command line you use to start your Java command line.

### Installing the ODBC Driver on Mac OS X

You can obtain the Vertica ODBC driver for Mac OS X as a .pkg file through the myVertica portal. You can run the installer as a regular Mac OS X installer or silently. This driver is compatible with both 32-bit and 64-bit applications.

The installer is designed to be used with the standard iODBC Driver Manager included in Mac OS X. While Mac OS X ships with the iODBC Driver Manager already installed, you may choose to download the most recent version of the driver at the iODBC.org website.

By default, the installer installs the driver in the following location: `/Library/Vertica/ODBC/lib/libverticaodbc.dylib`. The installer also automatically registers a driver named "Vertica" with the iODBC Driver Manager.

To use the unixODBC Driver Manager instead of Apple's iODBC Driver Manager, see the unixODBC.org website.

### Before You Download the Driver

If you installed a previous version of the Vertica ODBC driver for Mac OS X, your system may already have a registered driver named "Vertica." In this case, if you must remove or rename the older version of the driver before installing the Vertica ODBC driver .pkg.

To have multiple versions of the driver installed on your system at the same time, you must rename the currently installed version of the driver to something other than "Vertica." You can do so using the Apple ODBC Administrator Tool.

To rename the driver:

1. Using your web browser, download and install the Apple ODBC Administrator Tool.

2. Locate and open the ODBC Administrator Tool after installation:
   a. Navigate to Finder > Applications > Utilities.
   b. Open the ODBC Administrator Tool.
3. Click the Drivers tab, and then select the driver named "Vertica."

4. Click the Configure button. A dialog box opens.
   a. In Description, enter a new name for the driver, and then click OK. The dialog box closes.
   b. On the ODBC Administrator page, click Apply.

5. Exit the ODBC Administrator Tool.

Download the Driver

Follow these steps to download the Vertica ODBC driver for Mac OS X:

1. On your Mac OS X client system, open a browser, and log in to the myVertica portal.

2. Install the Vertica ODBC driver for Mac OS X:
   a. Navigate to the Downloads tab, and scroll to the Client Software section.
   b. Click the download link for the Mac OS X ODBC installer.

3. Accept the license agreement, and wait for the download to complete.

Install the Mac OS X ODBC Driver

As a Mac OS X Administrator, double-click the installer to start the installation. Follow the prompts as the wizard guides you through each step of the process.

Note: After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see Creating an ODBC DSN for Macintosh OS X Clients.

Silently Install the Mac OS X ODBC Driver

1. Log into the client Mac in one of two ways:
   - As an administrator account, if you are installing the driver for system-wide use
   - As the user who needs to use the Vertica ODBC driver

2. Open a terminal window. In the Finder, click Applications > Utilities > Terminal.

3. Install the .pkg file containing the ODBC driver using the command:
In the preceding .pkg command, change the path to that of the downloaded file, if:

- You downloaded the driver .pkg file to a directory other than your Downloads directory.
- You downloaded the driver using another user account.

Note: After installing the ODBC driver, you must create a DSN to be able to connect to your Vertica database. For the procedure, see Creating an ODBC DSN for Macintosh OS X Clients.

Uninstall the Mac OS X ODBC Driver

Uninstalling the Mac OS X ODBC Client-Driver does not remove any existing DSNs associated with the driver.

To uninstall:

1. Open a terminal window.
2. Enter the command:

   ```bash
   sudo /Library/Vertica/ODBC/bin/Uninstall
   ```

Upgrade or Downgrade the Mac OS X ODBC Driver

All installations of the Vertica ODBC driver for Mac OS X are uniquely identified by a package ID and version number. The package ID does not change between versions, but the version number does. If you attempt multiple installations of the same version of the driver, a name collision error occurs. Therefore, multiple installations of the same version of the driver cannot coexist on a single operating system.

- **Upgrading**—Newly installed versions of the Vertica ODBC driver for Mac OS X automatically upgrade the relevant driver system settings. Any DSNs associated with a previous version of the driver are not affected, except that they begin using the newer version of the driver.

- **Downgrading**—Run the uninstall script to remove the current version of the Vertica ODBC driver for Mac OS X. Complete this step before installing an older driver version.
ODBC Driver Settings on Mac OS X

ODBC driver settings are automatically configured using the Vertica ODBC driver installer on Mac OS X. It is not necessary to configure additional ODBC driver settings on Mac OS X platforms beyond what is automatically configured by the installer. You can, however, set the ODBC driver settings by editing the VERTICAINI environment variable in each user's ~/.MacOSX/environment.plist file. See the Environment Variables entry in the Apple Developer's Library for more information.

See Additional Parameter Settings for a list of the additional settings.
Creating an ODBC Data Source Name (DSN)

A Data Source Name (DSN) is the logical name that is used by Open Database Connectivity (ODBC) to refer to the driver and other information that is required to access data from a data source. Whether you are developing your own ODBC client code or you are using a third-party tool that needs to access Vertica using ODBC, you need to configure and test a DSN. The method you use depends upon the client operating system you are using.

Refer to the following sections for information specific to your client operating system.

- Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX
- Creating an ODBC DSN for Windows Clients
- Creating an ODBC DSN for Macintosh OS X Clients
- Data Source Name (DSN) Connection Properties
- Setting DSN Connection Properties

Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX

You define DSN on Linux, Solaris, and other UNIX-like platforms in a text file. Your client's driver manager reads this file to determine how to connect to your Vertica database. The driver manager usually looks for the DSN definitions in two places:

- `/etc/odbc.ini`
- `~/.odbc.ini` (a file named `.odbc.ini` in the user's home directory)

Users must be able to read the `odbc.ini` file in order to use it to connect to the database. If you use a global `odbc.ini` file, consider creating a UNIX group with read access to the file. Then add the users who need to use the DSN to this group.

The structure of these files is the same, only their location differs. If both files are present, the `~/.odbc.ini` file usually overrides the system-wide `/etc/odbc.ini` file.

**Note:** See your ODBC driver manager's documentation for details on where these files should be located and any other requirements.
odbc.ini File Structure

The odbc.ini is a text file that contains two types of lines:

- Section definitions, which are text strings enclosed in square brackets.
- Parameter definitions, which contain a parameter name, an equals sign (=), and then the parameter's value.

The first section of the file is always named [ODBC Data Sources], and contains a list of all the DSNs that the odbc.ini file defines. The parameters in this section are the names of the DSNs, which appear as section definitions later in the file. The value is a text description of the DSN and has no function. For example, an odbc.ini file that defines a single DSN named VerticaDSN could have this ODBC Data Sources section:

```
[ODBC Data Sources]
HPVerticaDSN = "vmartdb"
```

Appearing after the ODBC data sources section are sections that define each DSN. The name of a DSN section must match one of the names defined in the ODBC Data Sources section.

Configuring the odbc.ini file:

To create or edit the DSN definition file:

1. Using the text editor of your choice, open odbc.ini or ~/.odbc.ini.

2. Create an ODBC Data Sources section and define a parameter:
   - Whose name is the name of the DSN you want to create
   - Whose value is a description of the DSN

   For example, to create a DSN named VMart, you would enter:

   ```
   [ODBC Data Sources]
   VMart = "VMart database on Vertica"
   ```

3. Create a section whose name matches the DSN name you defined in step 2. In this section, you add parameters that define the DSN's settings. The most commonly-defined parameters are:
Description – Additional information about the data source.

Driver – The location and designation of the Vertica ODBC driver, or the name of a driver defined in the odbcinst.ini file (see below). For future compatibility, use the name of the symbolic link in the library directory, rather than the library file:

- `/opt/vertica/lib`, on 32-bit clients
- `/opt/vertica/lib64`, on 64-bit clients

For example, the symbolic link for the 64-bit ODBC driver library is:

```
/opt/vertica/lib64/libverticaodbc.so
```

The symbolic link always points to the most up-to-date version of the Vertica client ODBC library. Use this link so that you do not need to update all of your DSNs when you update your client drivers.

Database – The name of the database running on the server. This example uses vmartdb for the vmartdb.

ServerName — The name of the server where Vertica is installed. Use localhost if Vertica is installed on the same machine.

You can provide an IPv4 address, IPv6 address, or host name.

In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.

UID — Either the database superuser (same name as database administrator account) or a user that the superuser has created and granted privileges. This example uses the user name dbadmin.

PWD — The password for the specified user name. This example leaves the password field blank.

Port — The port number on which Vertica listens for ODBC connections. For example, 5433.

ConnSettings — Can contain SQL commands separated by a semicolon. These commands can be run immediately after connecting to the server.
- **SSLKeyFile** — The file path and name of the client's private key. This file can reside anywhere on the system.

- **SSLCertFile** — The file path and name of the client's public certificate. This file can reside anywhere on the system.

- **Locale** — The default locale used for the session. By default, the locale for the database is: en_US@collation=binary (English as in the United States of America). Specify the locale as an ICU Locale. See the ICU User Guide (http://userguide.icu-project.org/locale) for a complete list of parameters that can be used to specify a locale.

- **PreferredAddressFamily**:  

  The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:  
  - ipv4—Connect to the server using IPv4.  
  - ipv6—Connect to the server using IPv6.  
  - none—Use the IP address provided by the DNS server.

For example:

```
[VMart]  
Description = Vmart Database  
Driver = /opt/vertica/lib64/libverticaodbc.so  
Database = vmartdb  
Servername = host01  
UID = dbadmin  
PwD =  
Port = 5433  
ConnSettings =  
AutoCommit = 0  
SSLKeyFile = /home/dbadmin/client.key  
SSLCertFile = /home/dbadmin/client.crt  
Locale = en_US@collation=binary
```

See [Data Source Name (DSN) Connection Properties](#) for a complete list of parameters including Vertica-specific ones.

### Using an odbcinst.ini File

Instead of giving the path of the ODBC driver library in your DSN definitions, you can use the name of a driver defined in the odbcinst.ini file. This method is useful method if you have many DSNs and often need to update them to point to new driver libraries. It also allows you to set some additional ODBC parameters, such as the threading model.
Just as in the odbc.ini file, odbcinst.ini has sections. Each section defines an ODBC driver that can be referenced in the odbc.ini files.

In a section, you can define the following parameters:

- **Description** — Additional information about the data source.

- **Driver** — The location and designation of the Vertica ODBC driver, such as /opt/vertica/lib64/libverticaodbc.so

For example:

```ini
[HPVertica]
Description = Vertica ODBC Driver
Driver = /opt/vertica/lib64/libverticaodbc.so
```

Then, in your odbc.ini file, use the name of the section you created in the odbcinst.ini file that describes the driver you want to use. For example:

```ini
[VMart]
Description = Vertica Vmart database
Driver = HPVertica
```

If you are using the unixODBC driver manager, you should also add an ODBC section to override its standard threading settings. By default, unixODBC serializes all SQL calls through ODBC, which prevents multiple parallel loads. To change this default behavior, add the following to your odbcinst.ini file:

```ini
[ODBC]
Threading = 1
```

### Configuring Additional ODBC Settings

On Linux and UNIX systems, you need to configure some additional driver settings before you can use your DSN. See [Required ODBC Driver Configuration Settings for Linux and UNIX](#) for details.

### Testing an ODBC DSN Using Isql

The unixODBC driver manager includes a utility named isql, which is a simple ODBC command-line client. It lets you to connect to a DSN to send commands and receive results, similarly to vsql.

To use isql to test a DSN connection:
1. Run the following command:

   ```bash
   $ isql -v DSNname
   ```

   Where `DSNname` is the name of the DSN you created.

   A connection message and a SQL prompt display. If they do not, you could have a configuration problem or you could be using the wrong user name or password.

2. Try a simple SQL statement. For example:

   ```sql
   SQL> SELECT table_name FROM tables;
   ```

   The isql tool returns the results of your SQL statement.

   **Note:** If you have not set the ErrorMessagesPath in the additional driver configuration settings, any errors during testing will trigger a missing error message file ("The error message NoSQLGetPrivateProfileString could not be found in the en-US locale"). See Required ODBC Driver Configuration Settings for Linux and UNIX for more information.

---

Creating an ODBC DSN for Windows Clients

To create a DSN for Microsoft Windows clients, you must perform the following tasks:

**Setting Up an ODBC DSN**

A *Data Source Name (DSN)* is the ODBC logical name for the drive and other information the database needs to access data. The name is used by Internet Information Services (IIS) for a connection to an ODBC data source.

This section describes how to use the Vertica ODBC Driver to set up an ODBC DSN. This topic assumes that the driver is already installed, as described in Installing Client Drivers on Windows.

To set up a DSN

1. Open the ODBC Administrator. For example, you could navigate to Start > Control Panel > Administrative Tools > Data Sources (ODBC).
2. Decide if you want all users on your client system to be able to access to the DSN for the Vertica database.

   - If you want all users to have access, then click the System DSN tab.
   - Otherwise, click the User DSN tab to create a DSN that is only usable by your Windows user account.

3. Click Add to create a new DSN to connect to the Vertica database.

4. Scroll through the list of drivers in the Create a New Data Source dialog box to locate the Vertica driver. Select the driver, and then click Finish.

   Note: If you have installed more than one version of the Vertica client drivers on your Windows client system, you may see multiple versions of the driver in this list. Choose the version that you know is compatible with your client application and Vertica Analytic Database server. If you are unsure, use the latest version of the driver.

The Vertica ODBC DSN configuration dialog box appears.

5. Click the More >>> button to view a description of the field you are editing and the connection string defined by the DSN.

6. Enter the information for your DSN. The following fields are required:

   - **DSN Name** — The name for the DSN. Clients use this name to identify the DSN to which they want to connect. The DSN name must satisfy the following requirements:
     - Its maximum length is 32 characters.
     - It is composed of ASCII characters except for the following: [ ] { } , ; ? * = ! @ 
     - It contains no spaces.

   - **Server** — The host name or IP address of the Vertica server to which you want to connect. Use localhost, if Vertica is installed on the same machine.

     You can provide an IPv4 address, IPv6 address, or host name.
In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the `PreferredAddressFamily` option to force the connection to use either IPv4 or IPv6.

The `PreferredAddressFamily` option is available on the Client Settings tab.

- **Backup Servers** — A comma-separated list of host names or IP addresses used to connect to if the server specified by the Server field is down. Optional.

- **Database** — The name of the Vertica database.

- **User Name** — The name of the user account to use when connecting to the database. If the application does not supply its own user name when connecting to the DSN, this account name is used to log into the database.

The rest of the fields are optional. See [DSN Parameters](#) for detailed information about the DSN parameters you can define.

7. If you want to test your connection:

   a. Enter at least a valid DSN name, Server name, Database, and either User name or select Windows authentication.

   b. If you have not selected Windows authentication, you can enter a password in the Password box. Alternately, you can select Password prompt to have the driver prompt you for a password when connecting.

   c. Click Test Connection.

8. When you have finished editing and testing the DSN, click OK. The Vertica ODBC DSN configuration window closes, and your new DSN is listed in the ODBC Data Source Administrator window.

9. Click OK to close the ODBC Data Source Administrator.

After creating the DSN, you can test it using Microsoft Excel 2007.

Setting up a 32-Bit DSN on 64-Bit Versions of Microsoft Windows

On 64-bit versions of Windows, the default ODBC Data Source Administrator creates and edits DSNs that are associated with the 64-bit Vertica ODBC library.
Attempting to use these 64-bit DSNs with a 32-bit client application results in an architecture mismatch error. Instead, you must create a specific 32-bit DSN for 32-bit clients by running the 32-bit ODBC Administrator usually located at:

c:\Windows\SysWOW64\odbcad32.exe

This administrator window edits a set of DSNs that are associated with the 32-bit ODBC library. You can then use your 32-bit client applications with the DSNs you create with this version of the ODBC administrator.

Encrypting Passwords on ODBC DSN

When you install an ODBC driver and create a Data Source Name (DSN) the DSN settings are stored in the registry, including the password. Encrypting passwords on ODBC DSN applies only to Windows systems.

Encrypting passwords on an ODBC data source name (DSN) provides security against unauthorized database access. The password is not encrypted by default and is stored in plain-text.

Note: Password encryption applies only to new or modified ODBC DSNs. If you have a DSN created in Version 8.0 or earlier and upgrade to 8.1, the password does not get encrypted regardless of the encryption settings.

Enable Password Encryption

Use the EncryptPassword parameter to enable or disable password encryption for an ODBC DSN:

- EncryptPassword = true enables password encryption
- EncryptPassword = false (default) disables password encryption

Set EncryptPassword in the Windows registry - HKEY_LOCAL_MACHINE > Software > Vertica > ODBC > Driver EncryptPassword=<true/false>.

Note: For 32 bit driver running on 64 bit windows verify password encryption here:

HKEY_LOCAL_MACHINE > Software > Wow6432Node > Vertica > ODBC > Driver > EncryptPassword=<true/false>

Encrypted passwords get updated in the following registry locations:

For a user DSN:
HKEY_CURRENT_USER -> Software -> ODBC -> ODBC.INI -> DSNNAMES -> PWD

For a system DSN:
HKEY_LOCAL_MACHINE -> Software -> ODBC -> ODBC.INI -> DSNNAMES -> PWD

Verify Password Encryption

Use Windows Registry editor to determine if password encryption is enabled based on the value of EncryptPassword. Depending on the type of DSN you installed, check the following:

For a user DSN: HKEY_CURRENT_USER > Software > ODBC > ODBC.INI > dsn name > isPasswordEncrypted=<1/0>

For a system DSN: HKEY_LOCAL_MACHINE > Software > ODBC > ODBC.INI > dsn name > isPasswordEncrypted=<1/0>

For each DSN, the value of the isPasswordEncrypted parameter indicates the status of the password encryption, where 1 indicates an encrypted password and 0 indicates an unencrypted password.

Testing an ODBC DSN Using Excel

You can use Microsoft Excel to verify that an application can connect to an ODBC data source or other ODBC application.

1. Open Microsoft Excel, and select Data > Get External Data > From Other Sources > From Microsoft Query.

2. When the Choose Data Source dialog box opens:
   a. Select New Data Source, and click OK.

   b. Enter the name of the data source.

   c. Select the Vertica driver.

   d. Click Connect.

3. When the Vertica Connection Dialog box opens, enter the connection information for the DSN, and click OK.
4. Click OK on the Create New Data Source dialog box to return to the Choose Data Source dialog box.

5. Select VMart_Schema*, and verify that the Use the Query Wizard check box is deselected. Click OK.

6. When the Add Tables dialog box opens, click Close.

7. When the Microsoft Query window opens, click the SQL button.

8. In the SQL window, write any simple query to test your connection. For example:

   ```sql
   SELECT DISTINCT calendar_year FROM date_dimension;
   ```

9. If you see the caution, "SQL Query can't be represented graphically. Continue anyway?" click OK.

   - The data values 2003, 2004, 2005, 2006, 2007 indicate that you successfully connected to and ran a query through ODBC.

10. Select File > Return Data to Microsoft Office Excel.

11. In the Import Data dialog box, click OK.

    The data is now available for use in an Excel worksheet.

---

### Creating an ODBC DSN for Macintosh OS X Clients

You can use the Vertica ODBC Driver to set up an ODBC DSN. This procedure assumes that the driver is already installed, as described in [Installing the ODBC Driver on Macintosh OS X](#).

---

### Setting Up a DSN

1. Using your web browser, download and install the Apple ODBC Administrator Tool.

2. Locate and open the ODBC Administrator Tool after installation:

   a. Navigate to Finder > Applications > Utilities.

   b. Open the ODBC Administrator Tool.
3. Click the Drivers tab, and verify that the Vertica driver is installed.

4. Specify if you want all users on your client system to be able to access the DSN for the Vertica database:
   - If you want all users to have access, then click the System DSN tab.
   - Otherwise, click the User DSN tab to create a DSN that is only usable by your Macintosh user account.

5. Click Add... to create a new DSN to connect to the Vertica database.

6. Scroll through the list of drivers in the Choose A Driver dialog box to locate the Vertica driver. Select the driver, and then click OK. A dialog box opens that requests DSN parameter information.

7. In the dialog box, enter the Data Source Name (DSN) and an optional Description. To do so, click Add to insert keywords (parameters) and values that define the settings needed to connect to your database, including database name, server host, database user name (such as dbadamin), database password, and port. Then, click OK.
8. In the ODBC Administrator dialog box, click Apply.

See Data Source Name (DSN) Connection Properties for a complete list of parameters including those specific to Vertica.

After configuring the ODBC Administrator Tool, you may need to configure additional driver settings before you can use your DSN, depending on your environment. See Additional ODBC Driver Configuration Settings for details.

Note: If you want to test your connection, use the iodbc test utility. For the procedure, see Testing a DSN Using iodbc test.

Testing an ODBC DSN Using iodbc test

The standard iODBC Driver Manager on OS X includes a utility named iodbc test that lets you test a DSN to verify that it is correctly configured. You pass this command a connection string in the same format that you would use to open an ODBC database connection. After configuring your DSN connection, you can run a query to verify that the connection works.

For example:

```
# iodbc test "DSN=VerticaDSN;UID=dbadmin;PWD=password"

iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.0607.1008
Driver: 07.01.0200 (verticaodbcw.so)
SQL> SELECT table_name FROM tables;
table_name
-----------------------------------------------
customer_dimension
product_dimension
promotion_dimension
date_dimension
vendor_dimension
employee_dimension
shipping_dimension
warehouse_dimension
inventory_fact
store_dimension
store_sales_fact
store_orders_fact
online_page_dimension
call_center_dimension
online_sales_fact
numbers
result set 1 returned 16 rows.
```
Data Source Name (DSN) Connection Properties

The following tables list the connection properties you can set in the DSNs for use with Vertica's ODBC driver.

Required Connection Properties

These connection properties are the minimum required to create a functioning DSN.

NOTE: If you use a host name (Servername) whose DNS entry resolves to multiple IP addresses, the client attempts to connect to the first IP address returned by the DNS. If a connection cannot be made to the first address, the client attempts to connect to the second, then the third, continuing until it either connects successfully or runs out of addresses.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>The file path and name of the driver used.</td>
<td>none</td>
</tr>
<tr>
<td>Database</td>
<td>The name of the database running on the server.</td>
<td>none</td>
</tr>
<tr>
<td>Servername</td>
<td>The host name or IP address of any active node in a Vertica cluster.</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>You can provide an IPv4 address, IPv6 address, or host name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In mixed IPv4/IPv6 networks, the DNS server configuration determines which</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IP version address is sent first. Use the PreferredAddressFamily option to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>force the connection to use either IPv4 or IPv6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>You can also use the aliases &quot;server&quot; and &quot;host&quot; for this property.</td>
<td></td>
</tr>
<tr>
<td>UID</td>
<td>The database username.</td>
<td>none</td>
</tr>
</tbody>
</table>

Optional Properties

These are basic properties that are optional.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>The port number on which Vertica listens for ODBC connections.</td>
<td>5433</td>
</tr>
<tr>
<td>PWD</td>
<td>The password for the specified user name. You may insert an empty string to leave this property blank.</td>
<td>none (login only succeeds if the user does not have a password set)</td>
</tr>
</tbody>
</table>
| PreferredAddressFamily | The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:  
- ipv4—Connect to the server using IPv4.  
- ipv6—Connect to the server using IPv6.  
- none—Use the IP address provided by the DNS server. | none          |
| Protocol          | Specifies the front-end protocol that the ODBC client driver uses to communicate with a previous Vertica server version. For the 9.0.x ODBC client driver to connect to Vertica server version 7.1, specify Protocol=3.5. Specifying the protocol, while not required, allows full backwards compatibility. | 3.6           |

Note: The default protocol version, 3.6, applies to Vertica server versions 7.2, 8.0, and 9.0.x. You do not need to specify the protocol property when connecting from an 9.0.x ODBC client driver to server versions 7.2, 8.0, and 9.0.x.
## Advanced Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>A Boolean value that controls whether the driver automatically commits transactions after executing a DML statement.</td>
<td>true</td>
</tr>
<tr>
<td>BackupServerNode</td>
<td>A string containing the host name or IP address that client libraries can try to connect to if the host specified in ServerName is unreachable. Connection attempts continue until successful or until the list of server nodes is exhausted. <strong>Valid values:</strong> Comma-separated list of servers optionally followed by a colon and port number.</td>
<td>none</td>
</tr>
<tr>
<td>ConnectionLoadBalance</td>
<td>A Boolean value that indicates whether the connection can be redirected to a host in the database other than the ServerNode. This affects the connection only if the load balancing is set to something other than &quot;none&quot;. When the node differs from the node the</td>
<td>false</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>client is connected to, the client disconnects and reconnects to the targeted node. See About Native Connection Load Balancing in the Administration Guide.</td>
<td></td>
</tr>
<tr>
<td>ConnSettings</td>
<td>A string containing SQL commands that the driver should execute immediately after connecting to the server. You can use this property to configure the connection, such as setting a schema search path. Reserved symbol: In the connection string ';', is a reserved symbol. To set multiple properties as part of ConnSettings properties, use '%3B' for ','. Also use '+', for spaces.</td>
<td>none</td>
</tr>
<tr>
<td>ConvertSquareBracketIdentifiers</td>
<td>Controls whether square-bracket query identifiers are converted to a double quote identifier for compatibility when making queries to a Vertica database.</td>
<td>false</td>
</tr>
<tr>
<td>DirectBatchInsert</td>
<td>A Boolean value that controls where data inserted through the connection is stored. When set to true, Vertica</td>
<td>false</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>directly inserts data into ROS containers. Otherwise, it stores data using AUTO mode. When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</td>
<td></td>
</tr>
</tbody>
</table>
| DriverStringConversions | Controls whether the ODBC driver performs type conversions on strings sent between the ODBC driver and the database. Possible values are:  
- NONE—No conversion in either direction. This results in the highest performance.  
- INPUT—Strings sent from the client to the server are converted, but strings sent from the server to the client are not.  
- OUTPUT—Strings sent by the server to the client are converted, but strings sent from the client to the server are not. | OUTPUT  |
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>**BOTH—**Strings are converted in both directions.</td>
<td>Locale: The locale used for the session. Specify the locale as an ICU Locale. See the ICU User Guide for a complete list of properties that can be used to specify a locale.</td>
<td>en_US@collation=binary (English as in the United States of America)</td>
</tr>
<tr>
<td></td>
<td>PromptOnNoPassword: [Windows only] Controls whether users are prompted to enter a password, if none is supplied by the connection string or DSN used to connect to Vertica. See Prompting Windows Users for Passwords.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>ReadOnly: A true or false value that controls whether the connection can read data only from Vertica.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>ResultBufferSize: Size of memory buffer for the large result sets in streaming mode. A value of 0 means ResultBufferSize is turned off. This property was previously called MaxMemoryCache.</td>
<td>131072 (128KB)</td>
</tr>
<tr>
<td></td>
<td>TransactionIsolation: Sets the transaction isolation for the server.</td>
<td>Server Default</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>connection. Valid values are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Read Committed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Serializable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Server Default</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Identification**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Standard/Vertica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Description for the DSN entry. <strong>Required? No</strong> Insert an empty string to leave the description empty.</td>
<td>none</td>
<td>Standard</td>
</tr>
<tr>
<td>Label / SessionLabel</td>
<td>Sets a label for the connection on the server. This value appears in the session_id column of the V_MONITOR SESSIONS system table. Label and SessionLabel are synonyms and can be used interchangeably.</td>
<td>none</td>
<td>Vertica</td>
</tr>
</tbody>
</table>

**Encryption**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Standard/Vertica</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLMode</td>
<td>Controls whether the connection to the database</td>
<td>prefer</td>
<td>Vertica</td>
</tr>
</tbody>
</table>
uses SSL encryption. Valid values follow. For descriptions of these values, refer to Configuring SSL for ODBC Clients.

- require
- prefer—Prefers that the server use SSL. If the server does not offer an encrypted channel, the client requests one. The first connection attempt to the database tries to use SSL. If that attempt fails, a second connection is attempted over a clear channel.
- allow—Makes a connection to the server whether the server uses SSL or not. The first connection attempt to the database is attempted over a clear channel. If that fails, a second connection is attempted over SSL.
- disable—Never connects to the server using SSL. Typically, you use this setting for troubleshooting.

SSLCertFile
The absolute path of the client's public certificate file. This file can reside anywhere on the system.

SSLKeyFile
The absolute path to the client's private key file. This file can reside anywhere on the system.

Third-Party Compatibility

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Standard/Vertica</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnsAsChar</td>
<td>Specifies how character column types are reported when the driver is in Unicode</td>
<td>false</td>
<td>Vertica</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
<td>Standard/Vertica</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>mode. When set to false, the ODBC driver reports the data type of character columns as WCHAR. If you set ColumnsAsChar to true, the driver identifies character column as CHAR. You typically use this setting for compatibility with some third-party clients, such as Informatica.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThreePartNaming</td>
<td>A Boolean value that controls how catalog names are interpreted by the driver. When this value is false, the driver reports that catalog names are not supported. When catalog names are not supported, they cannot be used as a filter in database metadata API calls. In this case, the driver returns NULL as the catalog name in all driver metadata results. When this value is true, catalog names can be used as a filter in database metadata API calls. In this case, the driver returns the</td>
<td>false (UNIX)</td>
<td>Vertica</td>
</tr>
<tr>
<td></td>
<td>A Boolean value that controls how catalog names are interpreted by the driver. When this value is false, the driver reports that catalog names are not supported. When catalog names are not supported, they cannot be used as a filter in database metadata API calls. In this case, the driver returns NULL as the catalog name in all driver metadata results. When this value is true, catalog names can be used as a filter in database metadata API calls. In this case, the driver returns the</td>
<td>true (Window)</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
<td>Standard/Vertica</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>database name as the catalog name in metadata results. Some third-party applications assume a certain catalog behavior and do not work properly with the default values. Enable this option if your client software expects to get the catalog name from the database metadata and use it as part of a three-part name reference.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnforceBatchInsertNullConstraints</td>
<td>Prevents NULL values from being loaded into columns with a NOT NULL constraint during batch inserts. When this value is set to true, batch inserts roll back when NULL values are inserted into columns with NOT NULL constraints. When this value is set to false, batch insert behavior is unchanged. Vertica recommends only using this property with SAP Data Services as it could negatively impact database performance.</td>
<td>false</td>
<td>Vertica</td>
</tr>
</tbody>
</table>
Kerberos Connection Properties

Use the following properties for client authentication using Kerberos.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Standard/Vertica</th>
</tr>
</thead>
<tbody>
<tr>
<td>KerberosServiceName</td>
<td>Provides the service name portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM</td>
<td>vertica</td>
<td>Vertica</td>
</tr>
<tr>
<td>KerberosHostname</td>
<td>Provides the instance or hostname portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM</td>
<td>Value specified in the servername connection string property</td>
<td>Vertica</td>
</tr>
</tbody>
</table>

See Also

- Required ODBC Driver Configuration Settings for Linux and UNIX

Setting DSN Connection Properties

The properties in the following tables are common for all user and system DSN entries. The examples provided are for Windows clients.

To edit DSN properties:

- On UNIX and Linux client platforms, you can edit the odbc.ini file. The location of this file is specific to the driver manager. See Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX.

- On Windows client platforms, you can edit some DSN properties using the Vertica ODBC client driver interface. See Creating an ODBC DSN for Windows Clients.

- You can also edit the DSN properties directly by opening the DSN entry in the Windows registry (for example, at HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\DSNname).
Directly editing the registry can be risky, so you should only use this method for properties that cannot be set through the ODBC driver's user interface, or via your client code.

- You can set properties in the connection string when opening a connection using the `SQLDriverConnect()` function:

  ```
  sqlRet = SQLDriverConnect(sql_hDBC, 0, (SQLCHAR*)"DSN=DSNName;Locale=en_GB@collation=binary", SQL_NTS, szDNS, 1024,&nSize, SQL_DRIVER_NOPROMPT);
  ```

  **Note:** In the connection string ';' is a reserved symbol. If you need to set multiple properties as part of the ConnSettings property use '%3B' in place of ';'. Also use '+"' instead of spaces.

  For example:

  ```
  sqlRet = SQLDriverConnect(sql_hDBC, 0, (SQLCHAR*)"DSN=VerticaSQL;ConnSettings=set+search_path+to+a,b,c%3Bset+locale=ch;SSLMode=prefer", SQL_NTS, szDNS, 1024,&nSize, SQL_DRIVER_NOPROMPT);
  ```

- Your client code can retrieve DSN property values after a connection has been made to Vertica using the `SQLGetConnectAttr()` and `SQLGetStmtAttr()` API calls. Some properties can be set and using `SQLSetConnectAttr()` and `SQLSetStmtAttr()`.

  For details of the list of properties specific to Vertica see ODBC Header Files specific to Vertica.
Programming ODBC Client Applications

Vertica provides an Open Database Connectivity (ODBC) driver that allows applications to connect to the Vertica database. This driver can be used by custom-written client applications that use the ODBC API to interact with Vertica. ODBC is also used by many third-party applications to connect to Vertica, including business intelligence applications and extract, transform, and load (ETL) applications.

This section details the process for configuring the Vertica ODBC driver. It also explains how to use the ODBC API to connect to Vertica in your own client applications.

This section assumes that you have already installed the ODBC libraries on your client system. If you have not, see Client Drivers.

ODBC Architecture

The ODBC architecture has four layers:

- **Client Application**
  Is an application that opens a data source through a Data Source Name (DSN). It then sends requests to the data source, and receives the results of those requests. Requests are made in the form of calls to ODBC functions.

- **Driver Manager**
  Is a library on the client system that acts as an intermediary between a client application and one or more drivers. The driver manager:
    - Resolves the DSN provided by the client application.
    - Loads the driver required to access the specific database defined within the DSN.
    - Processes ODBC function calls from the client or passing them to the driver.
    - Retrieves results from the driver.
    - Unloads drivers when they are no longer needed.

On Windows and Mac client systems, the driver manager is provided by the operating system. On Linux and UNIX systems, you usually need to install a driver manager. See ODBC...
**Prerequisites** for a list of driver managers that can be used with Vertica on your client platform.

- **Driver**
  
  A library on the client system that provides access to a specific database. It translates requests into the format expected by the database, and translates results back into the format required by the client application.

- **Database**
  
  The database processes requests initiated at the client application and returns results.

**ODBC Feature Support**

The ODBC driver for Vertica supports the most of the features defined in the Microsoft ODBC 3.5 specifications. The following features are *not* supported:

- Updatable result sets
- Backwards scrolling cursors
- Cursor attributes
- More than one open statement per connection. For example you cannot execute a new statement while another statement has a result set open. If you need to execute multiple statements at once, open multiple database connections.

- Keysets
- Bookmarks

The Vertica ODBC driver accurately reports its capabilities. If you need to determine whether it complies with a specific feature, you should query the driver's capabilities directly using the SQLGetInfo() function.

**Vertica and ODBC Data Type Translation**

Most data types are transparently converted between Vertica and ODBC. This section explains several data types require special handling.
<table>
<thead>
<tr>
<th>Vertica Data Types</th>
<th>C Data Type</th>
<th>ODBC C Typedef</th>
<th>C Type Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY, VARBINARY</td>
<td>char[]</td>
<td>SQL_BINARY</td>
<td>SQL_C_BINARY</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>char[]</td>
<td>SQL_LONGVARBINARY</td>
<td>SQL_C_BINARY</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SQLSMALLINT</td>
<td>SQL_SMALLINT</td>
<td>SQL_C_SSHORT</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>char[]</td>
<td>SQL_CHAR</td>
<td>SQL_C_CHAR</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>char[]</td>
<td>SQL_LONGVARCHAR</td>
<td>SQL_C_CHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_DATE_STRUCT</td>
<td>SQL_DATE_TYPE</td>
<td>SQL_C_DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>SQL_TIME_STRUCT</td>
<td>SQL_TIME_TYPE</td>
<td>SQL_C_TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>SQL_TIMESTAMP_STRUCT</td>
<td>SQL_TIMESTAMP_TYPE</td>
<td>SQL_C_TIMESTAMP</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>SQL_INTERVAL_STRUCT</td>
<td>SQL_INTERVAL_DAY_TO_SECOND</td>
<td>SQL_C_INTERVAL_DAY_TO_SECOND</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>SQL_INTERVAL_STRUCT</td>
<td>SQL_INTERVAL_DAY_TO_SECOND</td>
<td>SQL_C_INTERVAL_DAY_TO_SECOND</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>SQL_INTERVAL_STRUCT</td>
<td>SQL_INTERVAL_YEAR_TO_MONTH</td>
<td>SQL_C_INTERVAL_YEAR_TO_MONTH</td>
</tr>
<tr>
<td>DOUBLE PRECISION FLOAT</td>
<td>SQLREAL</td>
<td>SQL_REAL</td>
<td>SQL_C_FLOAT</td>
</tr>
<tr>
<td>INTEGER, BIGINT, SMALLINT</td>
<td>SQLBIGINT</td>
<td>SQL_BIGINT</td>
<td>SQL_C_SBIGINT</td>
</tr>
<tr>
<td>NUMERIC, DECIMAL, NUMBER, MONEY</td>
<td>SQL_NUMERIC_STRUCT</td>
<td>SQL_NUMERIC</td>
<td>SQL_C_NUMERIC</td>
</tr>
<tr>
<td>GEOMETRY</td>
<td>char[]</td>
<td>SQL_LONGVARBINARY</td>
<td>SQL_C_CHAR</td>
</tr>
<tr>
<td>Vertica Data Types</td>
<td>C Data Type</td>
<td>ODBC C Typedef</td>
<td>C Type Identifier</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>GEOGRAPHY</td>
<td>char[]</td>
<td>SQL_LONGVARCHAR</td>
<td>SQL_C_CHAR</td>
</tr>
<tr>
<td>UUID</td>
<td>SQLGUID (see note below)</td>
<td>SQL_GUID</td>
<td>SQL_C_GUID</td>
</tr>
</tbody>
</table>

**Notes**

- The GEOMETRY and GEOGRAPHY data types are treated as LONG VARCHAR data by the ODBC driver.

- Vertica supports the standard interval data types supported by ODBC. See [Interval Data Types](http://www.microsoft.com) in Microsoft's ODBC reference.

- Vertica version 9.0.0 introduced the UUID data type, including JDBC support for UUIDs. The Vertica ADO.NET, ODBC, and OLE DB clients added full support for UUIDs in version 9.0.1. Vertica maintains backwards compatibility with the earlier client driver versions that do not support the UUID data type (see [Client Driver and Server Version Compatibility](http://www.vertica.com) for the versions of the client drivers that Vertica supports). When one of these older clients queries tables with UUID columns in them, Vertica automatically translates the native UUID values to CHAR values. Also, when an older client inserts data into a UUID column, Vertica automatically converts the CHAR value sent by the client into a native UUID value. Vertica also reports the data type of these columns as CHAR when an older client queries a UUID column's metadata.

**See Also**

- [SQL Data Types](http://www.vertica.com)

- [Using LONG VARCHAR and LONG VARBINARY Data Types with ODBC](http://www.vertica.com)

- [Using GEOMETRY and GEOGRAPHY Data Types in ODBC](http://www.vertica.com)

- [SQL Data Types](http://www.vertica.com) in the Microsoft ODBC reference documentation
Using LONG VARCHAR and LONG VARBINARY Data Types with ODBC

The ODBC drivers support the LONG VARCHAR and LONG VARBINARY data types similarly to VARCHAR and VARBINARY data types. When binding input or output parameters to a LONG VARCHAR or LONG VARBINARY column in a query, use the SQL_LONGVARCHAR and SQL_LONGVARBINARY constants to set the column's data type. For example, to bind an input parameter to a LONG VARCHAR column, you would use a statement that looks like this:

```c
rc = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_LONGVARCHAR, 80000, 0, (SQLPOINTER)myLongString, sizeof(myLongString), NULL);
```

**Note:** Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).

ODBC Header File Specific to Vertica

The Vertica ODBC driver provides a C header file named verticaodbc.h that defines several useful constants that you can use in your applications. These constants let you access and alter settings specific to Vertica.

This file's location depends on your client operating system:

- `/opt/vertica/include` on Linux and UNIX systems.
- `C:\Program Files (x86)\Vertica\ODBC\include` on Windows systems.

The constants defined in this file are listed below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Associated Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_ATTR_undefined_variable Vertica.DBMS_UPPERCASE]].RESULT_BUFFER_SIZE</td>
<td>Sets the size of the buffer used when retrieving results from</td>
<td>SQLSetConnectAttr() SQLGetConnectAttr()</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Associated Function</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>SQL_ATTR_[[Undefined variable Vertica.DBMS-uppercase]]_DIRECT_BATCH_INSERT</td>
<td>Determines whether a batch is inserted directly into the ROS (1) or using AUTO mode (0). By default batches are inserted using AUTO. When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</td>
<td>SQLSetConnectAttr() SQLSetStmtAttr() SQLGetConnectAttr() SQLGetStmtAttr()</td>
</tr>
<tr>
<td>SQL_ATTR_[[Undefined variable Vertica.DBMS-uppercase]]_LOCALE</td>
<td>Changes the locale from en_US@collation= binary to the ICU locale specified. See Setting the Locale for</td>
<td>SQLSetConnectAttr() SQLGetConnectAttr()</td>
</tr>
</tbody>
</table>
Connecting to the Database

The first step in any ODBC application is to connect to the database. When you create the connection to a data source using ODBC, you use the name of the DSN that contains the details of the driver to use, the database host, and other basic information about connecting to the data source.

There are 4 steps your application needs to take to connect to a database:

1. Call `SQLAllocHandle()` to allocate a handle for the ODBC environment. This handle is used to create connection objects and to set application-wide settings.

2. Use the environment handle to set the version of ODBC that your application wants to use. This ensures that the data source knows which API your application will use to interact with it.

3. Allocate a database connection handle by calling `SQLAllocHandle()`. This handle represents a connection to a specific data source.

4. Use the `SQLConnect()` or `SQLDriverConnect()` functions to open the connection to the database.

   **Note:** If you specify a locale either in the connection string or in the DSN, the call to the connection function returns `SQL_SUCCESS_WITH_INFO` on a successful connection, with messages about the state of the locale.

When creating the connection to the database, use `SQLConnect()` when the only options you need to set at connection time is the username and password. Use `SQLDriverConnect()` when you want to change connection options, such as the locale.

The following example demonstrates connecting to a database using a DSN named ExampleDB. After it creates the connection successfully, this example simply closes it.

```c
// Demonstrate connecting to Vertica using ODBC. 
// Standard i/o library
```
```c
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>

int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hd1Env; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC 3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application version to ODBC 3.\n");
    }

    // Allocate a database handle.
    SQLHDBC hd1Dbc;
    ret = SQLAllocHandle(SQL_HANDLEDBC, hd1Env, &hd1Dbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated Database handle.\n");
    }

    // Connect to the database using
    // SQL Connect
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char* userID = "ExampleUser";
    const char* passwd = "password123";
    ret = SQLConnect(hd1Dbc, (SQLCHAR*)dsnName, 
                     SQL_NTS,(SQLCHAR*)userID,SQL_NTS, 
                     (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }

    // We're connected. You can do real
    // work here

    // When done, free all of the handles to close them
"
// in an orderly fashion.
printf("Disconnecting and freeing handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting from database. Transaction still open?\n");
    exit(EXIT_FAILURE);
}

SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);

Running the above code prints the following:
Allocated an environment handle.
Set application version to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Disconnecting and freeing handles.

See Setting the Locale for ODBC Sessions for an example of using SQLDriverConnect to connect to the database.

Notes

- If you use the DataDirect® driver manager, you should always use the SQL_DRIVER_NOPROMPT value for the SQLDriverConnect function's DriverCompletion parameter (the final parameter in the function call) when connecting to Vertica. Vertica's ODBC driver on Linux and UNIX platforms does not contain a UI, and therefore cannot prompt users for a password.
- On Windows client platforms, the ODBC driver can prompt users for connection information. See Prompting Windows Users for Missing Connection Properties for more information.
- If your database does not comply with your Vertica license agreement, your application receives a warning message in the return value of the SQLConnect() function. Always have your application examine this return value to see if it is SQL_SUCCESS_WITH_INFO. If it is, have your application extract and display the message to the user.
Enabling Native Connection Load Balancing in ODBC

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen.

If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See About Native Connection Load Balancing in the Administrator's Guide for details.

To enable native load balancing on your client, set the ConnectionLoadBalance connection parameter to true either in the DSN entry or in the connection string. The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the V_MONITOR.CURRENT_SESSION system table.

```c++
// Demonstrate enabling native load connection balancing.
// Standard i/o library
#include <stdlib.h>
#include <iostream>
#include <assert.h>
// Only needed for Windows clients
#include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>

using namespace std;
int main()
{
    SQLRETURN ret;  // Stores return value from ODBC API calls
    SQLHENV hd1Env; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    assert(SQL_SUCCEEDED(ret));

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    assert(SQL_SUCCEEDED(ret));
```
// Allocate a database handle.
SQLHDBC hd1Dbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
assert(SQL_SUCCEEDED(ret));

// Connect four times. If load balancing is on, client should
// connect to different nodes.
for (int x=1; x <= 4; x++) {
    // Connect to the database using SQLDriverConnect. Set
    // ConnectionLoadBalance to 1 (true) to enable load
    // balancing.
    cout << endl << "Connection attempt #" << x << ". . . ";
    const char *connStr = "DSN=VMart;ConnectionLoadBalance=1;"
                        "UID=ExampleUser;PWD=password123";

    ret = SQLDriverConnect(hd1Dbc, NULL, (SQLCHAR*)connStr, SQL_NTS,
                           NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
    if(!SQL_SUCCEEDED(ret)) {
        cout << "failed. Exiting." << endl;
        exit(EXIT_FAILURE);
    } else {
        cout << "succeeded" << endl;
    }
    // We're connected. Query the v_monitor.current_session table to
    // find the name of the node we've connected to.

    // Set up a statement handle
    SQLHSTMT hd1Stmt;
    SQLAllocHandle(SQL_HANDLE_STMT, hd1Dbc, &hd1Stmt);
    assert(SQL_SUCCEEDED(ret));

    ret = SQLExecDirect( hd1Stmt, (SQLCHAR*)"SELECT node_name FROM 
                         "V_MONITOR.CURRENT_SESSION;", SQL_NTS );
    if(SQL_SUCCEEDED(ret)) {
        // Bind variable to column in result set.
        SQLCHAR node_name[256];
        ret = SQLBindCol(hd1Stmt, 1, SQL_C_TCHAR, (SQLPOINTER)node_name,
                         sizeof(node_name), NULL);
        while(SQL_SUCCEEDED(ret = SQLFetchScroll(hd1Stmt, SQL_FETCH_NEXT,1))) {
            // Print the bound variables, which now contain the values from the
            // fetched row.
            cout << "Connected to node " << node_name << endl;
        }
    }
    // Free statement handle
    SQLFreeHandle(SQL_HANDLE_STMT, hd1Stmt);
    cout << "Disconnecting." << endl;
    ret = SQLDisconnect( hd1Dbc );
    assert(SQL_SUCCEEDED(ret));
}
// When done, free all of the handles to close them
// in an orderly fashion.
cout << endl << "Disconnecting." << endl;
SQLDisconnect( hd1Dbc );
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
cout << "Done!" << endl;
exit(EXIT_SUCCESS);
Running the above example produces output similar to the following:

```plaintext
Connection attempt #1... succeeded
Connected to node v_vmart_node0001
Disconnecting.

Connection attempt #2... succeeded
Connected to node v_vmart_node0003
Disconnecting.

Connection attempt #3... succeeded
Connected to node v_vmart_node0002
Disconnecting.

Connection attempt #4... succeeded
Connected to node v_vmart_node0001
Disconnecting.

Freeing handles...
Done!
```

**ODBC Connection Failover**

If a client application attempts to connect to a host in the Vertica Analytic Database cluster that is down, the connection attempt fails when using the default connection configuration. This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytic Database's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host specified in the connection settings is unreachable. It gives you two ways to let the client driver automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- Configure your DNS server to return multiple IP addresses for a host name. When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.

- Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.
For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts. Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

Choosing a Failover Method

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

**Note:** If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytic Database cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytic Database automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytic Database cluster. This may not be possible in your organization.

Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytic Database cluster.

Using DNS Failover

To use DNS failover, you need to change your DNS server’s settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytic Database cluster. You then have all client applications use this host name to connect to Vertica Analytic Database.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.
Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the BackupServerNode parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the BackupServerNode connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```cpp
// Demonstrate using connection failover.
// Standard i/o library
#include <stdlib.h>
#include <iostream>
#include <assert.h>

// Only needed for Windows clients
// #include &lt;windows.h&gt;

// SQL include files that define data types and ODBC API
// functions
#include &lt;sql.h&gt;
#include &lt;sqlext.h&gt;
#include &lt;sqltypes.h&gt;
using namespace std;

int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hd1Env; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    assert(SQL_SUCCEEDED(ret));

    // Set the ODBC version we are going to use to
    // 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    assert(SQL_SUCCEEDED(ret));

    // Allocate a database handle.
    SQLHDBC hd1Dbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
    assert(SQL_SUCCEEDED(ret));

    /* DSN for this connection specifies a bad node, and good backup nodes:
    [VMartBadNode]
    Description=VMart Vertica Database
    Driver=/opt/vertica/lib64/libverticaodbc.so */
```
Database=VMart
Servername=badnode.example.com
BackupServerNode=v_vmart_node0002.example.com,v_vmart_node0003.example.com

/*

// Connect to the database using SQLConnect
cout &lt;&lt; "Connecting to database." &lt;&lt; endl;
const char *dsnName = "VMartBadNode"; // Name of the DSN
const char* userID = "ExampleUser"; // Username
const char* passwd = "password123"; // password
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
    SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
    (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    cout &lt;&lt; "Could not connect to database." &lt;&lt; endl;
    exit(EXIT_FAILURE);
} else {
    cout &lt;&lt; "Connected to database." &lt;&lt; endl;
}
// We're connected. Query the v_monitor.current_session table to
// find the name of the node we've connected to.

// Set up a statement handle
SQLHSTMT hd1Stmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
assert(SQL_SUCCEEDED(ret));
ret = SQLExecDirect( hd1Stmt, (SQLCHAR*)"SELECT node_name FROM 
    "v_monitor.current_session;", SQL_NTS );
if(SQL_SUCCEEDED(ret)) {
    // Bind varible to column in result set.
    SQLTCHAR node_name[256];
    ret = SQLBindCol(hd1Stmt, 1, SQL_C_TCHAR, (SQLPOINTER)node_name,
        sizeof(node_name), NULL);
    while(SQL_SUCCEEDED(ret = SQLFetchScroll(hd1Stmt, SQL_FETCH_NEXT,1))) {
        // Print the bound variables, which now contain the values from the
        // fetched row.
        cout &lt;&lt; "Connected to node " &lt;&lt; &lt;&lt; node_name &lt;&lt; endl;
    }
}
cout &lt;&lt; "Disconnecting." &lt;&lt; endl;
ret = SQLDisconnect( hd1Dbc );
assert(SQL_SUCCEEDED(ret));

// When done, free all of the handles to close them
// in an orderly fashion.
cout &lt;&lt; endl &lt;&lt; "Freeing handles..." &lt;&lt; endl;
SQLFreeHandle(SQL_HANDLE_STMT,hd1Stmt);
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
cout &lt;&lt; "Done!" &lt;&lt; endl;
exit(EXIT_SUCCESS);
*/

When run, the example's output on the system console is similar to the following:
Connecting to database.
Connected to database.
Connected to node v_vmart_node0002
Disconnecting.

Freeing handles...
Done!

Notice that the connection was made to the first node in the backup list (node 2).

**Note:** When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See [Enabling Native Connection Load Balancing in ODBC](#).

### Prompting Windows Users for Missing Connection Properties

The Vertica Windows ODBC driver can prompt the user for connection information if required information is missing. The driver displays the Vertica Connection Dialog if the client application calls `SQLDriverConnect` to connect to Vertica and either of the following is true:

- the DriverCompletion property is set to `SQL_DRIVER_PROMPT`.
- the DriverCompletion property is set to `SQL_DRIVER_COMPLETE` or `SQL_DRIVER_COMPLETE_REQUIRED` and the connection string or DSN being used to connect is missing the server, database, or port information.

If either of the above conditions are true, the driver displays a Vertica Connection Dialog to the user to prompt for connection information.
The dialog has all of the property values supplied in the connection string or DSN filled in.

**Note:** Your connection string at least needs to specify Vertica as the driver, otherwise Windows will not know to use the Vertica ODBC driver to try to open the connection.

The required fields on the connection dialog are Database, UID, Server, and Port. Once these are filled in, the form enables the OK button.

If the user clicks Cancel on the dialog, the `SQLDriverConnect` function call returns `SQL_NO_DATA` immediately, without attempting to connect to Vertica. If the user supplies incomplete or incorrect information for the connection, the connection function returns `SQL_ERROR` after the connection attempt fails.

**Note:** If the DriverCompletion property of the `SQLDriverConnect` function call is `SQL_DRIVER_NOPROMPT`, the ODBC driver immediately returns a `SQL_ERROR` indicating that it cannot connect because not enough information has been supplied and the driver is not allowed to prompt the user for the missing information.

### Prompting Windows Users for Passwords

If the connection string or DSN supplied to the `SQLDriverConnect` function that client applications call to connect to Vertica lacks any of the required connection properties needed to connect, the Vertica’s Windows ODBC driver opens a dialog box to prompt the user to enter the missing information (see [Prompting Windows Users for Missing Connection Properties](#)). The user’s password is not normally considered a required connection property, since Vertica user accounts may not have a password. If the password property is missing, the ODBC driver still tries to connect to Vertica without supplying a password.
You can use the `PromptOnNoPassword` DSN parameter to force ODBC driver to treat the password as a required connection property. This parameter is useful if you do not want to store passwords in DSN entries. Passwords saved in DSN entries are insecure, since they are stored as clear text in the Windows registry and therefore visible to other users on the same system.

There are two other factors which also decide whether the ODBC driver displays the Vertica Connection Dialog. These are (in order of priority):

- The `SQLDriverConnect` function call's `DriverCompletion` parameter.
- Whether the DSN or connection string contain a password

The following table shows how the `PromptOnNoPassword` DSN parameter, the `DriverCompletion` parameter of the `SQLDriverConnect` function, and whether the DSN or connection string contains a password interact to control whether the Vertica Connection dialog appears.

<table>
<thead>
<tr>
<th>PromptOnNoPassword Setting</th>
<th>DriverCompletion Value</th>
<th>DSN or Connection String Contains Password?</th>
<th>Vertica Connection Dialog Displays?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>any value</td>
<td>SQL_DRIVER_PROMPT</td>
<td>any case</td>
<td>Yes</td>
<td>This DriverCompletion value forces the dialog to always appear, even if all required connection properties are supplied.</td>
</tr>
<tr>
<td>any value</td>
<td>SQL_DRIVER_NOPROMPT</td>
<td>any case</td>
<td>No</td>
<td>This DriverCompletion value always prevents the dialog from appearing.</td>
</tr>
<tr>
<td>any value</td>
<td>SQL_DRIVER_COMPLETE</td>
<td>Yes</td>
<td>No</td>
<td>Connection dialog displays if</td>
</tr>
</tbody>
</table>

Vertica Documentation
Connecting to Vertica

Vertica Analytic Database (9.0.x)
<table>
<thead>
<tr>
<th>PromptOnNoPassword Setting</th>
<th>DriverCompletion Value</th>
<th>DSN or Connection String Contains Password?</th>
<th>Vertica Connection Dialog Displays?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>another required connection property is missing.</td>
</tr>
<tr>
<td>true</td>
<td>SQL_DRIVER_COMPLETE</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>false (default)</td>
<td>SQL_DRIVER_COMPLETE</td>
<td>No</td>
<td>No</td>
<td>Connection dialog displays if another required connection property is missing.</td>
</tr>
</tbody>
</table>

The following example code demonstrates using the PromptOnNoPassword DSN parameter along with a system DSN.

```c
wstring connectString = L"DSN=VerticaDSN;PromptOnNoPassword=1;";
retcode = SQLDriverConnect(
    hdc, 0,
    (SQLCHAR*)connectString.c_str(),
    connectString.length(),
    OutConnStr, 255,
    &OutConnStrLen,
    SQL_DRIVER_COMPLETE );
```

**No Password Entry vs. Empty Passwords**

There is a difference between not having a password property in the connection string or DSN and having an empty password. The PromptOnNoPassword DSN parameter only has an effect if the connection string or DSN does not have a PWD property (which holds the user's password). If it does, even if it is empty, PromptOnNoPassword will not prompt the Windows ODBC driver to display the Vertica Connection Dialog.

This difference can cause confusion if you are using a DSN to provide the properties for your connection. Once you enter a password for a DSN connection in the Windows ODBC Manager
and save it, Windows adds a PWD property to the DSN definition in the registry. If you later delete the password, the PWD property remains in the DSN definition—value is just set to an empty string. The PWD property is created even if you just use the Test button on the ODBC Manager dialog to test the DSN and later clear it before saving the DSN.

Once the password has been set, the only way to remove the PWD property from the DSN definition is to delete it using the Windows Registry Editor:

1. On the Windows Start menu, click Run.
2. In the Run dialog, type regedit, then click OK.
3. In the Registry Editor window, click Edit > Find (or press Ctrl+F).
4. In the Find window, enter the name of the DSN whose PWD property you want to delete and click OK.
5. If find operation did not locate a folder under the ODBC.INI folder, click Edit > Find Next (or press F3) until the folder matching your DSN's name is highlighted.
6. Select the PWD entry and press Delete.
7. Click Yes to confirm deleting the value.

The DSN now does not have a PWD property and can trigger the connection dialog to appear when used along with PromptOnNoPassword=true and DriverConnect=SQL_DRIVER_COMPLETE.
Setting the Locale for ODBC Sessions

Vertica provides three ways to set the locale for an ODBC session:

- **Specify the locale for all connections made using the DSN:**
  - On Linux and other UNIX-like platforms: Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX
  - On Windows platforms, set the locale in the ODBC DSN configuration editor's Locale field on the Server Settings tab. See Creating an ODBC DSN for Windows Clients for detailed information.

- **Set the Locale connection parameter in the connection string in SQLDriverConnect() function.**
  
  For example:

  ```
  SQLDriverConnect(conn, NULL, (SQLCHAR*)"DSN=Vertica;Locale=en_GB@collation=binary", SQL_NTS, szConnOut, sizeof(szConnOut), &iAvailable, SQL_DRIVER_NOPROMPT)
  ```

- **Use the SQLSetConnectAttr() method with the SQL_ATTR_VERTICA_LOCALE constant and specify the ICU string as the attribute value.** See the example below.

**Notes**

- Having the client system use a non-Unicode locale (such as setting LANG=C on Linux platforms) and using a Unicode locale for the connection to Vertica can result in errors such as "(10170) String data right truncation on data from data source." If data received from Vertica isn't in UTF-8 format. The driver allocates string memory based on the system's locale setting, and non-UTF-8 data can trigger an overrun. You can avoid these errors by always using a Unicode locale on the client system.

  If you specify a locale either in the connection string or in the DSN, the call to the connection function returns SQL_SUCCESS_WITH_INFO on a successful connection, with messages about the state of the locale.

- **ODBC applications can be in either ANSI or Unicode mode:**
If Unicode, the encoding used by ODBC is UCS-2.

If ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 on the database server.

The ODBC driver converts UCS-2 to UTF-8 when passing to the Vertica server and converts data sent by the Vertica server from UTF-8 to UCS-2.

- If the end-user application is not already in UCS-2, the application is responsible for converting the input data to UCS-2, or unexpected results could occur. For example:
  - On non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
  - Or the symbol provided in the alternate encoding could be a valid UCS-2 symbol; in this case, incorrect data is inserted into the database.

ODBC applications should set the correct server session locale using SQLSetConnectAttr (if different from database-wide setting) in order to set the proper collation and string functions behavior on server.

The following example code demonstrates setting the locale using both the connection string and through the SQLSetConnectAttr() function.

```c
// Standard i/o library
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// SQL include files that define data types and ODBC API
// functions
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
// Vertica-specific definitions. This include file is located as
// /opt/vertica/include on database hosts.
#include <verticaodbc.h>
int main()
{
    SQLRETURN ret; // Stores return value from ODBC API calls
    SQLHENV hd1Env; // Handle for the SQL environment object
    // Allocate an a SQL environment object
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Set the ODBC version we are going to use to 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
```
(SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not set application version to ODBC 3.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Set application version to ODBC 3.\n");
}

// Allocate a database handle.
SQLHDBC hdlDbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated Database handle.\n");
}

// Connect to the database using SQLDriverConnect
printf("Connecting to database.\n");
// Set the locale to English in Great Britain.
const char *connStr = "DSN=ExampleDB;locale=en_GB;"
    "UID=dbadmin;PWD=password123";
ret = SQLDriverConnect(hdlDbc, NULL, (SQLCHAR*)connStr, SQL_NTSC
    NULL, 0, NULL, SQL_DRIVER_NOPROMPT );
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Set the locale to a new value
const char* newLocale = "en_GB";
SQLSetConnectAttr(hdlDbc, SQL_ATTRIB VERTICA_LOCAL, (SQLCHAR*)newLocale,
    SQL_NTSC);

// Get the Locale
char locale[256];
SQLGetConnectAttr(hdlDbc, SQL_ATTRIB VERTICA_LOCAL, locale, sizeof(locale),
    0);
printf("Locale is set to: %s\n", locale);
// When done, free all of the handles to close them
// in an orderly fashion.
printf("Disconnecting and freeing handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting from database. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
AUTOCOMMIT and ODBC Transactions

The AUTOCOMMIT connection attribute controls whether INSERT, ALTER, COPY and other data-manipulation statements are automatically committed after they complete. By default, AUTOCOMMIT is enabled—all statements are committed after they execute. This is often not the best setting to use, since it is less efficient. Also, you often want to control whether a set of statements are committed as a whole, rather than have each individual statement committed. For example, you may only want to commit a series of inserts if all of the inserts succeed. With AUTOCOMMIT disabled, you can roll back the transaction if one of the statements fail.

If AUTOCOMMIT is on, the results of statements are committed immediately after they are executed. You cannot roll back a statement executed in AUTOCOMMIT mode.

For example, when AUTOCOMMIT is on, the following single INSERT statement is automatically committed:

```c
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,"
    "'Smith, Sam', '123-456-789');", SQL_NTS);
```

If AUTOCOMMIT is off, you need to manually commit the transaction after executing a statement. For example:

```c
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,"
    "'Smith, Sam', '123-456-789');", SQL_NTS);
// Other inserts and data manipulations
// Commit the statements(s)
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
```

The inserted row is only committed when you call SQLEndTran(). You can roll back the INSERT and other statements at any point before committing the transaction.

**Note**: Prepared statements cache the AUTOCOMMIT setting when you create them using SQLPrepare(). Later changing the connection's AUTOCOMMIT setting has no effect on the AUTOCOMMIT settings of previously created prepared statements. See [Using Prepared Statements](#) for details.

The following example demonstrates turning off AUTOCOMMIT, executing an insert, then manually committing the transaction.

```c
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
```
# include <sql.h>
# include <sqltypes.h>
# include <sqlext.h>

int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLHDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated Database handle.\n");
    }
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName, SQL_NTS,
                     (SQLCHAR*)userID,SQL_NTS,
                     (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }
    // Get the AUTOCOMMIT state
    SQLINTEGER autoCommitState;
    SQLGetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, &autoCommitState, 0, NULL);
    printf("Autocommit is set to: %d\n", autoCommitState);

    // Disable AUTOCOMMIT
    printf("Disabling autocommit.\n");
    ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF,
                            SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not disable autocommit.\n");
        exit(EXIT_FAILURE);
    }
}
}  

// Get the AUTOCOMMIT state again
SQLGetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, &autoCommitState, 0, NULL);
printf("Autocommit is set to: %d\n", autoCommitState);

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create a table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers", SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
  "(CustID int, CustName varchar(100), Phone_Number char(15))");,
SQL_NTS);

// Insert a single row.
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"INSERT INTO customers VALUES(500,
  '"Smith, Sam', '123-456-789'"), SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not perform single insert.\n");
} else {
    printf("Performed single insert.\n");
}

// Need to commit the transaction before closing, since autocommit is // disabled. Otherwise SQLDisconnect returns an error.
printf("Committing transaction.\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Error committing transaction.\n");
    exit(EXIT_FAILURE);
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect(hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting from database. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);

Running the above code results in the following output:

Allocated an environment handle.
Set application to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Autocommit is set to: 1
Disabling autocommit.
Autocommit is set to: 0
Performed single insert.
Committing transaction.
Free handles.

Note: You can also disable AUTOCOMMIT in the ODBC connection string. See Setting DSN Connection Properties for more information.

Retrieving Data Through ODBC

To retrieve data through ODBC, you execute a query that returns a result set (SELECT, for example), then retrieve the results using one of two methods:

- Use the SQLFetch() function to retrieve a row of the result set, then access column values in the row by calling SQLGetData().

- Use the SQLBindColumn() function to bind a variable or array to a column in the result set, then call SQLExtendedFetch() or SQLFetchScroll() to read a row of the result set and insert its values into the variable or array.

In both methods you loop through the result set until you either reach the end (signaled by the SQL_NO_DATA return status) or encounter an error.

Note: Vertica supports one cursor per connection. Attempting to use more than one cursor per connection will result in an error. For example, you receive an error if you execute a statement while another statement has a result set open.

The following code example demonstrates retrieving data from Vertica by:

1. Connecting to the database.
2. Executing a SELECT statement that returns the IDs and names of all tables.
3. Binds two variables to the two columns in the result set.
4. Loops through the result set, printing the ids and name values.

```c
// Demonstrate running a query and getting results by querying the tables
// system table for a list of all tables in the current schema.
// Some standard headers
#include <stdlib.h>
#include <sstream>
```
```cpp
#include <iostream>
#include <assert.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
// Use std namespace to make output easier using namespace std;
// Helper function to print SQL error messages.
template <typename HandleT>
void reportError(int handleTypeEnum, HandleT hdl)
{
    // Get the status records.
    SQLSMALLINT  i, MsgLen;
    SQLRETURN ret2;
    SQLCHAR     SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER   NativeError;
    i = 1;
cout << endl;
while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError,
                               Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {
    cout << "error record #" << i++ << endl;
    cout << "sqlstate: " << SqlState << endl;
    cout << "detailed msg: " << Msg << endl;
    cout << "native error code: " << NativeError << endl;
}
}

int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hd1Env;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    assert(SQL_SUCCEEDED(ret));
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
                         (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    assert(SQL_SUCCEEDED(ret));
    // Allocate a database handle.
    SQLHDBC hd1Dbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
    assert(SQL_SUCCEEDED(ret));
    // Connect to the database
    cout << "Connecting to database." << endl;
    const char* dsnName = "ExampleDB";
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hd1Dbc, (SQLCHAR*)dsnName,
                     SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
                     (SQLCHAR*)passwd, SQL_NTS);
    if(!SQL_SUCCEEDED(ret)) {
        cout << "Could not connect to database" << endl;
        reportError<SQLHDBC>(SQL_HANDLE_DBC, hd1Dbc);
        exit(EXIT_FAILURE);
    } else {
        cout << "Connected to database." << endl;
    }
    // Set up a statement handle
```

```c
SQLHSTMT hd1Stmt;
SQLAllocHandle(SQL_HANDLE_STMT, hd1Dbc, &hd1Stmt);
assert(SQL_SUCCEEDED(ret));

// Execute a query to get the names and IDs of all tables in the schema
// search p[ath (usually public).
ret = SQLExecDirect( hd1Stmt, (SQLCHAR*)"SELECT table_id, table_name "
    "FROM tables ORDER BY table_name", SQL_NTS);

if(!SQL_SUCCEEDED(ret)) {
    // Report error an go no further if statement failed.
    cout << "Error executing statement." << endl;
    reportError<SQLHDBC>(SQL_HANDLE_STMT, hd1Stmt);
    exit(EXIT_FAILURE);
} else {

    // Query succeeded, so bind two variables to the two columns in the
    // result set,
    cout << "Fetching results..." << endl;
    SQLBIGINT table_id; // Holds the ID of the table.
    SQLTCHAR table_name[256]; // buffer to hold name of table
    ret = SQLBindCol(hd1Stmt, 1, SQL_C_SBIGINT, (SQLPOINTER)&table_id,
        sizeof(table_id), NULL);
    ret = SQLBindCol(hd1Stmt, 2, SQL_C_TCHAR, (SQLPOINTER)table_name,
        sizeof(table_name), NULL);

    // Loop through the results,
    while( SQL_SUCCEEDED(ret = SQLFetchScroll(hd1Stmt, SQL_FETCH_NEXT,1))) {
        // Print the bound variables, which now contain the values from the
        // fetched row.
        cout << table_id << " | " << table_name << endl;
    }

    // See if loop exited for reasons other than running out of data
    if (ret != SQL_NO_DATA) {
        // Exit for a reason other than no more data... report the error.
        reportError<SQLHDBC>( SQL_HANDLE_STMT, hd1Stmt );
    }
}

// Clean up by shutting down the connection
cout << "Free handles." << endl;
ret = SQLDisconnect( hd1Dbc );
if(!SQL_SUCCEEDED(ret)) {
    cout << "Error disconnecting. Transaction still open?" << endl;
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hd1Stmt);
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
exit(EXIT_SUCCESS);
```

Running the example code in the vmart database produces output similar to this:
Connecting to database.
Connected to database.
Fetching results...

45035996273970908 | call_center_dimension
45035996273970836 | customer_dimension
45035996273972958 | customers
45035996273970848 | date_dimension
45035996273970856 | employee_dimension
45035996273970868 | inventory_fact
45035996273970804 | online_page_dimension
45035996273970912 | online_sales_fact
45035996273970840 | product_dimension
45035996273970844 | promotion_dimension
45035996273970860 | shipping_dimension
45035996273970876 | store_dimension
45035996273970894 | store_orders_fact
45035996273970880 | store_sales_fact
45035996273972806 | t
45035996273970852 | vendor_dimension
45035996273970864 | warehouse_dimension
Free handles.

Loading Data Through ODBC

A primary task for many client applications is loading data into the Vertica database. There are several different ways to insert data using ODBC, which are covered by the topics in this section.

Using a Single Row Insert

The easiest way to load data into Vertica is to run an INSERT SQL statement using the SQLExecuteDirect function. However, this method is limited to inserting a single row of data.

```sql
ret = SQLExecDirect(hstmt, (SQLTCHAR*)"INSERT into Customers values"
                   "(1,'abcda','efgh','1')", SQL_NTS);
```

Using Prepared Statements

Vertica supports using server-side prepared statements with both ODBC and JDBC. Prepared statements let you define a statement once, and then run it many times with different parameters. The statement you want to execute contains placeholders instead of parameters. When you execute the statement, you supply values for each placeholder.

Placeholders are represented by question marks (?) as in the following example query:

```sql
SELECT * FROM public.inventory_fact WHERE product_key = ?
```
Server-side prepared statements are useful for:

- Optimizing queries. Vertica only needs to parse the statement once.
- Preventing SQL injection attacks. A SQL injection attack occurs when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly run. Since a prepared statement is parsed separately from the input data, there is no chance the data can be accidentally executed by the database.
- Binding direct variables to return columns. By pointing to data structures, the code doesn't have to perform extra transformations.

The following example demonstrates a using a prepared statement for a single insert.

```c
// Some standard headers
#include <stdio.h>
#include <stdlib.h>

// Only needed for Windows clients
#include <windows.h>

// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

// Some constants for the size of the data to be inserted.
#define CUST_NAME_LEN 50
#define PHONE_NUM_LEN 15
#define NUM_ENTRIES 4

int main()
{
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hdlEnv;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }
    // Allocate a database handle.
    SQLDBC hdlDbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
```
const char* userID = "dbadmin";
const char* passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
    SQL_NTS, (SQLCHAR*)userID, SQL_NTS,
    (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Disable AUTOCOMMIT
printf("Disabling autocommit.\n");
ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not disable autocommit.\n");
    exit(EXIT_FAILURE);
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
    SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
    "(CustID int, CustName varchar(100), Phone_Number char(15));",
    SQL_NTS);

// Set up a bunch of variables to be bound to the statement
// parameters.

// Create the prepared statement. This will insert data into the
// table we created above.
printf("Creating prepared statement\n");
ret = SQLPrepare(hdlStmt, (SQLTCHAR*)"INSERT INTO customers (CustID, "
    "CustName, Phone_Number) VALUES(?,?,?)", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not create prepared statement\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
    exit(EXIT_FAILURE);
} else {
    printf("Created prepared statement.\n");
}
SQLINTEGER custID = 1234;
SQLCHAR custName[100] = "Fein, Fredrick";
SQLVARCHAR phoneNum[15] = "555-123-6789";
SQLLEN strFieldLen = SQL_NTS;
SQLLEN custIDLen = 0;
// Bind the data arrays to the parameters in the prepared SQL
// statement
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &custID, 0, &custIDLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custID array\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
}
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
exit(EXIT_FAILURE);
} else {
    printf("Bound custID to prepared statement\n");
}

// Bind CustNames
SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 50, 0, (SQLPOINTER)custName, 0, &strFieldLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custNames\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
    exit(EXIT_FAILURE);
} else {
    printf("Bound custName to prepared statement\n");
}

// Bind phoneNums
SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 15, 0, (SQLPOINTER)phoneNum, 0, &strFieldLen);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind phoneNums\n");
    SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
    SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
    SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNum to prepared statement\n");
}

// Execute the prepared statement.
printf("Running prepared statement...\n");
ret = SQLExecute(hdlStmt);
if(!SQL_SUCCEEDED(ret)) {
    printf("not successful!\n");
} else {
    printf("successful.\n");
}

// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hd1Dbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not commit transaction\n");
} else {
    printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hd1Dbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
exit(EXIT_SUCCESS);
}
Using Batch Inserts

You use batch inserts to insert chunks of data into the database. By breaking the data into batches, you can monitor the progress of the load by receiving information about any rejected rows after each batch is loaded. To perform a batch load through ODBC, you typically use a prepared statement with the parameters bound to arrays that contain the data to be loaded. For each batch, you load a new set of data into the arrays then execute the prepared statement.

When you perform a batch load, Vertica uses a COPY statement to load the data. Each additional batch you load uses the same COPY statement. The statement remains open until you end the transaction, close the cursor for the statement, or execute a non-INSERT statement.

Using a single COPY statement for multiple batches improves batch loading efficiency by:

- reducing the overhead of inserting individual batches
- combining individual batches into larger ROS containers

Note: If the database connection has AUTOCOMMIT enabled, then the transaction is automatically committed after each batch insert statement which closes the COPY statement. Leaving AUTOCOMMIT enabled makes your batch load much less efficient, and can cause added overhead in your database as all of the smaller loads are consolidated.

Even though Vertica uses a single COPY statement to insert multiple batches within a transaction, you can locate which (if any) rows were rejected due to invalid row formats or data type issues after each batch is loaded. See Finding the Number of Accepted Rows for details.

Note: While you can find rejected rows during the batch load transaction, other types of errors (such as running out of disk space or a node shutdown that makes the database unsafe) are only reported when the COPY statement ends.

Since the batch loads share a COPY statement, errors in one batch can cause earlier batches in the same transaction to be rolled back.

Batch Insert Steps

The steps your application needs to take in order to perform an ODBC Batch Insert are:
1. Connect to the database.

2. Disable autocommit for the connection.

3. Create a prepared statement that inserts the data you want to load.

4. Bind the parameters of the prepared statement to arrays that will contain the data you want to load.

5. Populate the arrays with the data for your batches.

6. Execute the prepared statement.

7. Optionally, check the results of the batch load to find rejected rows.

8. Repeat the previous three steps until all of the data you want to load is loaded.

9. Commit the transaction.

10. Optionally, check the results of the entire batch transaction.

The following example code demonstrates a simplified version of the above steps.

```c
#include <stdio.h>
#include <stdlib.h>

#include <windows.h>

#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

int main()
{
    int NUM_ENTRIES = 4;

    SQLRETURN ret;
    SQLHENV hd1Env;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }

    SQLHENV hd2Env;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, hd1Env, &hd2Env);
    if(!SQL_SUCCEEDED(ret)) {
        printf("Could not allocate a database handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated a database handle.\n");
    }

    SQLRETURN ret2;
    SQLHDBC db1Conn;
    ret2 = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &db1Conn);
    if(!SQL_SUCCEEDED(ret2)) {
        printf("Could not allocate a database connection.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated a database connection.\n");
    }

    SQLRETURN ret3;
} else {
    printf("Set application to ODBC 3.\n");
}

// Allocate a database handle.
SQLHDBC hdlDbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated Database handle.\n");
}

// Connect to the database
printf("Connecting to database.\n");
const char *dsnName = "ExampleDB";
const char* userID = "dbadmin";
const char* passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName, SQL_NTS, (SQLCHAR*)userID, SQL_NTS, (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Disable AUTOCOMMIT
printf("Disabling autocommit.\n");
ret = SQLSetConnectAttr(hdlDbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not disable autocommit.\n");
    exit(EXIT_FAILURE);
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create a table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
"(CustID int, CustName varchar(100), Phone_Number char(15));",
SQL_NTS);

// Create the prepared statement. This will insert data into the // table we created above.
printf("Creating prepared statement\n");
ret = SQLPrepare (hdlStmt, (SQLTCHAR*)"INSERT INTO customers (CustID, "
"CustName, Phone_Number) VALUES(?,?,?)", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not create prepared statement\n");
    exit(EXIT_FAILURE);
} else {
    printf("Created prepared statement.\n");
}

// This is the data to be inserted into the database.
SQLCHAR custNames[][50] = { "Allen, Anna", "Brown, Bill", "Chu, Cindy", "Dodd, Don"};
SQLINTEGER custIDs[] = { 100, 101, 102, 103};

// Bind the data arrays to the parameters in the prepared SQL statement. First is the custID.
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, (SQLPOINTER)custIDs, sizeof(SQLINTEGER), NULL);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custID array\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustIDs array to prepared statement\n");
}

// Bind CustNames
ret = SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 50, 0, (SQLPOINTER)custNames, 50, NULL);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind custNames\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound CustNames array to prepared statement\n");
}

// Bind phoneNums
ret = SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 15, 0, (SQLPOINTER)phoneNums, 15, NULL);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not bind phoneNums\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNums array to prepared statement\n");
}

// Tell the ODBC driver how many rows we have in the array.
ret = SQLSetStmtAttr(_hdlStmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)NUM_ENTRIES, 0);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not set parameter size\n");
    exit(EXIT_FAILURE);
} else {
    printf("Bound phoneNums array to prepared statement\n");
}

// Add multiple batches to the database. This just adds the same batch of data four times for simplicity's sake. Each call adds the 4 rows into the database.
for (int batchLoop=1; batchLoop<=5; batchLoop++) {
    // Execute the prepared statement, loading all of the data in the arrays.
    printf("Adding Batch #%d...", batchLoop);
    ret = SQLExecute(_hdlStmt);
    if(!SQL_SUCCEEDED(ret)) {
        printf("not successful!\n");
    } else {
        printf("successful.\n");
    }
}

// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hd1Dbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not commit transaction\n");
} else {
    printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect(hd1Dbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hd1stmt);
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
exit(EXIT_SUCCESS);

The result of running the above code is shown below.

Allocated an environment handle.
Set application to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Creating prepared statement
Created prepared statement.
Bound CustIDs array to prepared statement
Bound CustNames array to prepared statement
Bound phoneNums array to prepared statement
Adding Batch #1...successful.
Adding Batch #2...successful.
Adding Batch #3...successful.
Adding Batch #4...successful.
Adding Batch #5...successful.
Committing transaction
Committed transaction
Free handles.

The resulting table looks like this:

=> SELECT * FROM customers;

<table>
<thead>
<tr>
<th>CustID</th>
<th>CustName</th>
<th>Phone_Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Allen, Anna</td>
<td>1-617-555-1234</td>
</tr>
<tr>
<td>101</td>
<td>Brown, Bill</td>
<td>1-781-555-1212</td>
</tr>
<tr>
<td>102</td>
<td>Chu, Cindy</td>
<td>1-508-555-4321</td>
</tr>
<tr>
<td>103</td>
<td>Dodd, Don</td>
<td>1-617-555-4444</td>
</tr>
<tr>
<td>100</td>
<td>Allen, Anna</td>
<td>1-617-555-1234</td>
</tr>
<tr>
<td>101</td>
<td>Brown, Bill</td>
<td>1-781-555-1212</td>
</tr>
<tr>
<td>102</td>
<td>Chu, Cindy</td>
<td>1-508-555-4321</td>
</tr>
<tr>
<td>103</td>
<td>Dodd, Don</td>
<td>1-617-555-4444</td>
</tr>
<tr>
<td>100</td>
<td>Allen, Anna</td>
<td>1-617-555-1234</td>
</tr>
<tr>
<td>101</td>
<td>Brown, Bill</td>
<td>1-781-555-1212</td>
</tr>
<tr>
<td>102</td>
<td>Chu, Cindy</td>
<td>1-508-555-4321</td>
</tr>
</tbody>
</table>
Note: An input parameter bound with the SQL_C_NUMERIC data type uses the default numeric precision (37) and the default scale (0) instead of the precision and scale set by the SQL_NUMERIC_STRUCT input value. This behavior adheres to the ODBC standard. If you do not want to use the default precision and scale, use SQLSetDescField() or SQLSetDescRec() to change them in the statement's attributes.

Tracking Load Status (ODBC)

After loading a batch of data, your client application can get the number of rows that were processed and find out whether each row was accepted or rejected.
# Finding the Number of Accepted Rows

To get the number of rows processed by a batch, you add an attribute named SQL_ATTR_PARAMS_PROCESSED_PTR to the statement object that points to a variable to receive the number rows:

```c
SQLULEN rowsProcessed;
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &rowsProcessed, 0);
```

When your application calls SQLExecute() to insert the batch, the Vertica ODBC driver saves the number of rows that it processed (which is not necessarily the number of rows that were successfully inserted) in the variable you specified in the SQL_ATTR_PARAMS_PROCESSED_PTR statement attribute.
Finding the Accepted and Rejected Rows

Your application can also set a statement attribute named SQL_ATTR_PARAM_STATUS_PTR that points to an array where the ODBC driver can store the result of inserting each row:

```c
SQLUSMALLINT rowResults[NUM_ENTRIES];
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAM_STATUS_PTR, rowResults, 0);
```

This array must be at least as large as the number of rows being inserted in each batch.

When your application calls SQLExecute to insert a batch, the ODBC driver populates the array with values indicating whether each row was successfully inserted (SQL_PARAM_SUCCESS or SQL_PARAM_SUCCESS_WITH_INFO) or encountered an error (SQL_PARAM_ERROR).

The following example expands on the example shown in Using Batch Inserts to include reporting the number of rows processed and the status of each row inserted.

```c
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
#include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

// Helper function to print SQL error messages.
template<typename HandleT>
void reportError(int handleTypeEnum, HandleT hdl)
{
    // Get the status records.
    SQLUSMALLINT i, MsgLen;
    SQLRETURN ret2;
    SQLCHAR SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER NativeError;
    i = 1;
    printf("\n");
    while ((ret2 = SQLGetDiagRec(handleTypeEnum, hdl, i, SqlState, &NativeError, Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA) {
        printf("error record %d\n", i);
        printf("sqlstate: %s\n", SqlState);
        printf("detailed msg: %s\n", Msg);
        printf("native error code: %d\n\n", NativeError);
        i++;
    }
}
int main()
{
    // Number of data rows to insert
    
```
const int NUM_ENTRIES = 4;

SQLRETURN ret;
SQLHENV hdlEnv;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hdlEnv);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate a handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated an environment handle.\n");
}
ret = SQLSetEnvAttr(hdlEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not set application version to ODBC3.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Set application to ODBC 3.\n");
}
SQLHDBC hdlDbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hdlEnv, &hdlDbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated Database handle.\n");
}
// Connect to the database
printf("Connecting to database.\n");
const char *dsnName = "ExampleDB";
const char* userID = "dbadmin";
const char* passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName, SQL_NTS, (SQLCHAR*)userID, SQL_NTS, (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    reportError<SQLHDBC>(SQL_HANDLE_DBC, hdlDbc);
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}
// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers", SQL_NTS);
// Create a table into which we can store data
printf("Creating table.\n");
ret = SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers "
    "(CustID int, CustName varchar(50), Phone_Number char(15));", SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
} else {
    printf("Created table.\n");
}
// Create the prepared statement. This will insert data into the
// table we created above.
print("Creating prepared statement\n");
ret = SQLPrepare(hdlStmt, (SQLCHAR*)"INSERT INTO customers (CustID, "
  "CustName, Phone_Number) VALUES (?, ?, ?)", SQL_NT5);
if(!SQL_SUCCEEDED(ret)) {
  reportError<SQLHDBC>( SQL_HANDLE_STMT, hd1Stmt );
  exit(EXIT_FAILURE);
} else {
  printf("Created prepared statement.\n");
}

// This is the data to be inserted into the database.
char custNames[][50] = { "Allen, Anna", "Brown, Bill", "Chu, Cindy", 
  "Dodd, Don" };
SQLINTEGER custIDs[] = { 100, 101, 102, 103};
  "1-508-555-4321", "1-617-555-4444"};

// Bind the data arrays to the parameters in the prepared SQL
// statement
ret = SQLBindParameter(hdlStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 
  0, 0, (SQLPOINTER)custIds, sizeof(SQLINTEGER), NULL);
if(!SQL_SUCCEEDED(ret)) {
  reportError<SQLHDBC>( SQL_HANDLE_STMT, hd1Stmt );
  exit(EXIT_FAILURE);
} else {
  printf("Bound CustIDs array to prepared statement\n");
}

// Bind CustNames
SQLBindParameter(hdlStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR, 
  50, 0, (SQLPOINTER)custNames, 50, NULL);
if(!SQL_SUCCEEDED(ret)) {
  reportError<SQLHDBC>( SQL_HANDLE_STMT, hd1Stmt );
  exit(EXIT_FAILURE);
} else {
  printf("Bound CustNames array to prepared statement\n");
}

// Bind phoneNums
SQLBindParameter(hdlStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 
  15, 0, (SQLPOINTER)phoneNums, 15, NULL);
if(!SQL_SUCCEEDED(ret)) {
  reportError<SQLHDBC>( SQL_HANDLE_STMT, hd1Stmt );
  exit(EXIT_FAILURE);
} else {
  printf("Bound phoneNums array to prepared statement\n");
}

// Set up a variable to recieve number of parameters processed.
SQLULEN rowsProcessed;
// Set a statement attribute to point to the variable
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &rowsProcessed, 0);
// Set up an array to hold the result of each row insert
SQLUSMALLINT rowResults[NUM_ENTRIES];
// Set a statement attribute to point to the array
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAM_STATUS_PTR, rowResults, 0);
// Tell the ODBC driver how many rows we have in the
// array.
SQLSetStmtAttr(hdlStmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)NUM_ENTRIES, 0);
// Add multiple batches to the database. This just adds the same
// batch of data over and over again for simplicity's sake.
for (int batchLoop=1; batchLoop<=5; batchLoop++) {
  // Execute the prepared statement, loading all of the data
  // into the arrays.
printf("Adding Batch #%d...", batchLoop);
ret = SQLExecute(hdlStmt);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
    exit(EXIT_FAILURE);
}
// Number of rows processed is in rowsProcessed
printf("Params processed: %d\n", rowsProcessed);
printf("Results of inserting each row:\n");
int i;
for (i = 0; i<NUM_ENTRIES; i++) {
    SQLUSMALLINT result = rowResults[i];
    switch(rowResults[i]) {
    case SQL_PARAM_SUCCESS:
    case SQL_PARAM_SUCCESS_WITH_INFO:
        printf(" Row %d inserted successfully\n", i+1);
        break;
    case SQL_PARAM_ERROR:
        printf(" Row %d was not inserted due to an error.", i+1);
        break;
    default:
        printf(" Row %d had some issue with it: %d\n", i+1, result);
    }
}
// Done with batches, commit the transaction
printf("Commit Transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(!SQL_SUCCEEDED(ret)) {
    reportError<SQLHDBC>( SQL_HANDLE_STMT, hdlStmt );
}
// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hdlDbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}

Running the example code produces the following output:

Allocated an environment handle. Set application to ODBC 3.
Allocated Database handle.
Connecting to database.
Connected to database.
Creating table.
Created table.
Creating prepared statement
Created prepared statement.
Bound CustIDs array to prepared statement
Bound CustNames array to prepared statement
Bound phoneNums array to prepared statement
Adding Batch #1...Params processed: 4
Results of inserting each row:
   Row 1 inserted successfully
   Row 2 inserted successfully
   Row 3 inserted successfully
   Row 4 inserted successfully
Adding Batch #2...Params processed: 4
Results of inserting each row:
   Row 1 inserted successfully
   Row 2 inserted successfully
   Row 3 inserted successfully
   Row 4 inserted successfully
Adding Batch #3...Params processed: 4
Results of inserting each row:
   Row 1 inserted successfully
   Row 2 inserted successfully
   Row 3 inserted successfully
   Row 4 inserted successfully
Adding Batch #4...Params processed: 4
Results of inserting each row:
   Row 1 inserted successfully
   Row 2 inserted successfully
   Row 3 inserted successfully
   Row 4 inserted successfully
Adding Batch #5...Params processed: 4
Results of inserting each row:
   Row 1 inserted successfully
   Row 2 inserted successfully
   Row 3 inserted successfully
   Row 4 inserted successfully
Commit Transaction
Free handles.

Error Handling During Batch Loads

When loading individual batches, you can find information on how many rows were accepted and what rows were rejected (see Finding the Number of Accepted Rows for details). Other errors, such as disk space errors, do not occur while inserting individual batches. This behavior is caused by having a single COPY statement perform the loading of multiple consecutive batches. Using the single COPY statement makes the batch load process perform much faster. It is only when the COPY statement closes that the batched data is committed and Vertica reports other types of errors.

Your bulk loading application should check for errors when the COPY statement closes. Normally, you force the COPY statement to close by calling the SQLEndTran() function to end the transaction. You can also force the COPY statement to close by closing the cursor using the SQLCloseCursor() function, or by setting the database connection's AutoCommit property to true before inserting the last batch in the load.
**Note:** The COPY statement also closes if you execute any non-insert statement. However having to deal with errors from the COPY statement in what might be an otherwise-unrelated query is not intuitive, and can lead to confusion and a harder to maintain application. You should explicitly end the COPY statement at the end of your batch load and handle any errors at that time.

### Using the COPY Statement

The COPY statement lets you bulk load data from a file on stored on a database node into the Vertica database. This method is the most efficient way to load data into Vertica because the file resides on the database server. One drawback is that only a database superuser can use COPY, since it requires privilege in order to access the filesystem of the database node.

One drawback of using COPY instead of performing batch loads is that you can only get results of the load (the number of accepted and rejected rows) when the COPY statement has finished. With batch loads, you can monitor the progress as batches are inserted. The ability to monitor the progress of a load can be a useful feature if you want to stop loading if a large portion of the data is being rejected.

A primary concern when bulk loading data using COPY is deciding whether the data should be loaded directly into ROS using the DIRECT option, or by using the AUTO method (loading into WOS until it fills, then loading into ROS). You should load directly into the ROS when your transaction will load a large (more than 100MB of data or so) amount of data.

**Note:** The exceptions/rejections files are created on the client machine when the exceptions and rejected data modifiers are specified on the COPY command. Specify a local path and filename for these modifiers when executing a COPY query from the driver.

The following example loads data into the WOS (Write Optimized Store) until it fills, then stores additional data directly in ROS (Read Optimized Store).

```sql
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers"
"FROM '/data/customers.txt' AUTO",SQL_NTS);
```

The following example loads data into the ROS (Read Optimized Store).

```sql
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers"
"FROM '/data/customers.txt' DIRECT",SQL_NTS);
```

See the **COPY** statement in the SQL Reference Manual for more information about its syntax and use.

The following example demonstrates using the COPY command.
// Some standard headers
#include <stdio.h>
#include <stdlib.h>

// Only needed for Windows clients
#include <windows.h>

// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>

// Helper function to determine if an ODBC function call returned
// successfully.
bool notSuccess(SQLRETURN ret) {
    return (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO);
}

int main() {
    // Set up the ODBC environment
    SQLRETURN ret;
    SQLHENV hd1Env;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
    if(notSuccess(ret)) {
        printf("Could not allocate a handle.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Allocated an environment handle.\n");
    }
}

    // Tell ODBC that the application uses ODBC 3.
    ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
    if(notSuccess(ret)) {
        printf("Could not set application version to ODBC3.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Set application to ODBC 3.\n");
    }

    // Allocate a database handle.
    SQLHDBC hd1Dbc;
    ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
    // Connect to the database
    printf("Connecting to database.\n");
    const char *dsnName = "ExampleDB";
    const char* userID = "dbadmin";
    const char* passwd = "password123";
    ret = SQLConnect(hd1Dbc, (SQLCHAR*)dsnName,
            SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
            (SQLCHAR*)passwd, SQL_NTS);
    if(notSuccess(ret)) {
        printf("Could not connect to database.\n");
        exit(EXIT_FAILURE);
    } else {
        printf("Connected to database.\n");
    }

    // Disable AUTOCOMMIT
    printf("Disabling autocommit.\n");
    ret = SQLSetConnectAttr(hd1Dbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF, SQL_NTS);
    if(notSuccess(ret)) {
printf("Could not disable autocommit.
");
exit(EXIT_FAILURE);
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);
// Create table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers"
"(Last_Name char(50) NOT NULL, First_Name char(50), Email char(50), "
"Phone_Number char(15));",
SQL_NTS);
// Run the copy command to load data into ROS.
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers "
"FROM '/data/customers.txt' DIRECT",
SQL_NTS);
if(notSuccess(ret)) {
 printf("Data was not successfully loaded.
");
exit(EXIT_FAILURE);
} else {
 // Get number of rows added.
SQLLEN numRows;
ret=SQLRowCount(hdlStmt, &numRows);
printf("Successfully inserted %d rows.
", numRows);
}

// Done with batches, commit the transaction
printf("Committing transaction\n");
ret = SQLEndTran(SQL_HANDLE_DBC, hdlDbc, SQL_COMMIT);
if(notSuccess(ret)) {
 printf("Could not commit transaction\n");
} else {
 printf("Committed transaction\n");
}

// Clean up
printf("Free handles.\n");
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hdlDbc);
SQLFreeHandle(SQL_HANDLE_ENV, hdlEnv);
exit(EXIT_SUCCESS);
}

The example prints the following when run:

Allocated an environment handle.
Set application to ODBC 3.
Connecting to database.
Connected to database.
Disabling autocommit.
Successfully inserted 10001 rows.
Committing transaction
Committed transaction
Streaming Data From the Client Using COPY LOCAL

The LOCAL option of the SQL COPY statement lets you stream data from a file on a client system to your Vertica database. This statement works through the ODBC driver, making the task of transferring data files from the client to the server much easier.

The LOCAL option of COPY works transparently through the ODBC driver. Just have your client application execute a statement containing a COPY LOCAL statement, and the ODBC driver will read and stream the data file from the client to the server. For example:

```c
// Some standard headers
#include <stdio.h>
#include <stdlib.h>
// Only needed for Windows clients
// #include <windows.h>
// Standard ODBC headers
#include <sql.h>
#include <sqltypes.h>
#include <sqlext.h>
int main()
{
// Set up the ODBC environment
SQLRETURN ret;
SQLHENV hd1Env;
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hd1Env);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate a handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Allocated an environment handle.\n");
}
// Tell ODBC that the application uses ODBC 3.
ret = SQLSetEnvAttr(hd1Env, SQL_ATTR_ODBC_VERSION,
    (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UINTEGER);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not set application version to ODBC3.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Set application to ODBC 3.\n");
}
// Allocate a database handle.
SQLHDBC hd1Dbc;
ret = SQLAllocHandle(SQL_HANDLE_DBC, hd1Env, &hd1Dbc);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not allocate a database handle.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Set application to ODBC 3.\n");
}
// Connect to the database
printf("Connecting to database.\n");
```
const char *dsnName = "ExampleDB";
const char* userID = "dbadmin";
const char* passwd = "password123";
ret = SQLConnect(hdlDbc, (SQLCHAR*)dsnName,
                SQL_NTS,(SQLCHAR*)userID,SQL_NTS,
                (SQLCHAR*)passwd, SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Could not connect to database.\n");
    exit(EXIT_FAILURE);
} else {
    printf("Connected to database.\n");
}

// Set up a statement handle
SQLHSTMT hdlStmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdlDbc, &hdlStmt);

// Create table to hold the data
SQLExecDirect(hdlStmt, (SQLCHAR*)"DROP TABLE IF EXISTS customers",
             SQL_NTS);
SQLExecDirect(hdlStmt, (SQLCHAR*)"CREATE TABLE customers"
             "(Last_Name char(50) NOT NULL, First_Name char(50), Email char(50), "
             "Phone_Number char(15));",
             SQL_NTS);

// Run the copy command to load data into ROS.
ret=SQLExecDirect(hdlStmt, (SQLCHAR*)"COPY customers"
                   "FROM LOCAL '/home/dbadmin/customers.txt' DIRECT",
                   SQL_NTS);
if(!SQL_SUCCEEDED(ret)) {
    printf("Data was not successfully loaded.\n");
    exit(EXIT_FAILURE);
} else {
    // Get number of rows added.
    SQLLEN numRows;
    ret=SQLRowCount(hdlStmt, &numRows);
    printf("Successfully inserted %d rows.\n", numRows);
}

// COPY commits automatically, unless it is told not to, so
// there is no need to commit the transaction.

// Clean up
printf("Free handles.\n");
ret = SQLDisconnect( hd1Dbc );
if(!SQL_SUCCEEDED(ret)) {
    printf("Error disconnecting. Transaction still open?\n");
    exit(EXIT_FAILURE);
}
SQLFreeHandle(SQL_HANDLE_STMT, hdlStmt);
SQLFreeHandle(SQL_HANDLE_DBC, hd1Dbc);
SQLFreeHandle(SQL_HANDLE_ENV, hd1Env);
exit(EXIT_SUCCESS);
This example is essentially the same as the example shown in Using the COPY Statement, except it uses the COPY statement's LOCAL option to load data from the client system rather than from the filesystem of the database node.

Note: On Windows clients, the path you supply for the COPY LOCAL file is limited to 216 characters due to limitations in the Windows API.

See Also
- COPY LOCAL

Programming JDBC Client Applications

The Vertica JDBC driver provides you with a standard JDBC API. If you have accessed other databases using JDBC, you should find accessing Vertica familiar. This section explains how to use the JDBC to connect your Java application to Vertica.

You must first install the JDBC client driver on all client systems that will be accessing the Vertica database. For installation instructions, see Installing the Vertica Client Drivers.

For more information about JDBC:
- Vertica Analytic Database JDBC Documentation (Vertica extensions)
- An Introduction to JDBC, Part 1

JDBC Feature Support

The Vertica JDBC driver complies with the JDBC 4.0 standards (although it does not implement all of the optional features in them). Your application can use the DatabaseMetaData class to determine if the driver supports a particular feature it wants to use. In addition, the driver implements the Wrapper interface, which lets your client code discover Vertica-specific extensions to the JDBC standard classes, such as VerticaConnection and VerticaStatement classes.

Some important facts to keep in mind when using the Vertica JDBC driver:
• Cursors are forward only and are not scrollable. Result sets cannot be updated.

• A connection supports executing a single statement at any time. If you want to execute multiple statements simultaneously, you must open multiple connections.

• Because Vertica does not have stored procedures, CallableStatement is not supported. The DatabaseMetaData.getProcedures() and .getProcedureColumns() methods return information about SQL functions (including User Defined Functions) instead of stored procedures.

Multiple SQL Statement Support

The Vertica JDBC driver can execute strings containing multiple statements. For example:

```java
stmt.executeUpdate("CREATE TABLE t(a INT);INSERT INTO t VALUES(10);");
```

Only the Statement interface supports executing strings containing multiple SQL statements. You cannot use multiple statement strings with PreparedStatement. COPY statements that copy a file from a host file system work in a multiple statement string. However, client COPY statements (COPY FROM STDIN) do not work.

Multiple Batch Conversion to COPY Statements

The Vertica JDBC driver converts all batch inserts into Vertica COPY statements. If you turn off your JDBC connection's AutoCommit property, the JDBC driver uses a single COPY statement to load data from sequential batch inserts which can improve load performance by reducing overhead. See Batch Inserts Using JDBC Prepared Statements for details.

Multiple JDBC Version Support

The Vertica JDBC driver implements both JDBC 3.0 and JDBC 4.0 compliant interfaces. The interface that the driver returns to your application depends on the JVM version on which it is running. If your application is running on a 5.0 JVM, the driver supplies your application with JDBC 3.0 classes. If your application is running on a 6.0 or later JVM, the driver supplies it with JDBC 4.0 classes.
Multiple Active Result Sets (MARS)

The Vertica JDBC driver supports Multiple Active Result Sets (MARS). MARS allows the execution of multiple queries on a single connection. While ResultBufferSize sends the results of a query directly to the client, MARS stores the results first on the server. Once query execution has finished and all of the results have been stored, you can make a retrieval request to the server to have rows returned to the client.
Creating and Configuring a Connection

Before your Java application can interact with Vertica, it must create a connection. Connecting to Vertica using JDBC is similar to connecting to most other databases.

Importing SQL Packages

Before creating a connection, you must import the Java SQL packages. A simple way to do so is to import the entire package using a wildcard:

```java
import java.sql.*;
```

You may also want to import the `Properties` class. You can use an instance of this class to pass connection properties when instantiating a connection, rather than encoding everything within the connection string:

```java
import java.util.Properties;
```

If your application needs to run in a Java 5 JVM, it uses the older JDBC 3.0-compliant driver. This driver requires that you to manually load the Vertica JDBC driver using the `Class.forName()` method:

```java
// Only required for old JDBC 3.0 driver
try {
    Class.forName("com.vertica.jdbc.Driver");
} catch (ClassNotFoundException e) {
    // Could not find the driver class. Likely an issue
    // with finding the .jar file.
    System.err.println("Could not find the JDBC driver class.");
    e.printStackTrace();
    return; // Exit. Cannot do anything further.
}
```

Your application may run in a Java 6 or later JVM. If so, then the JVM automatically loads the Vertica JDBC 4.0-compatible driver without requiring the call to `Class.forName`. However, making this call does not adversely affect the process. Thus, if you want your application to be compatible with both Java 5 and Java 6 (or later) JVMs, it can still call `Class.forName`.

Opening the Connection

With SQL packages imported, you are ready to create your connection by calling the `DriverManager.getConnection()` method. You supply this method with at least the following information:
- The IP address or host name of a node in the database cluster:

  You can provide an IPv4 address, IPv6 address, or host name.

  In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.

- Port number for the database

- Name of the database

- Username of a database user account

- Password of the user (if the user has a password)

The first three parameters are always supplied as part of the connection string, a URL that tells the JDBC driver where to find the database. The format of the connection string is:

"jdbc:vertica://VerticaHost:portNumber/databaseName"

The first portion of the connection string selects the Vertica JDBC driver, followed by the location of the database.

You can provide the last two parameters, username and password, to the JDBC driver, in one of three ways:

- As part of the connection string. The parameters are encoded similarly to URL parameters:

  "jdbc:vertica://VerticaHost:portNumber/databaseName?user=username&password=password"

- As separate parameters to DriverManager.getConnection():

  ```java
  Connection conn = DriverManager.getConnection(
      "jdbc:vertica://VerticaHost:portNumber/databaseName",
      "username", "password");
  ```

- In a Properties object:

  ```java
  Properties myProp = new Properties();
  myProp.put("user", "username");
  myProp.put("password", "password");
  Connection conn = DriverManager.getConnection(
      "jdbc:vertica://VerticaHost:portNumber/databaseName", myProp);
  ```

Of these three methods, the Properties object is the most flexible because it makes passing additional connection properties to the getConnection() method easy. See Connection
Properties and Setting and Getting Connection Property Values for more information about the additional connection properties.

If there is any problem establishing a connection to the database, the getConnection() method throws a SQLException on one of its subclasses. To prevent an exception, enclose the method within a try-catch block, as shown in the following complete example of establishing a connection.

```java
import java.sql.*;
import java.util.Properties;

public class VerySimpleVerticaJDBCExample {
    public static void main(String[] args) {
        /*
         * If your client needs to run under a Java 5 JVM, it will use the older
         * JDBC 3.0-compliant driver, which requires you manually load the
         * driver using Class.forName
         *
         * try { Class.forName("com.vertica.jdbc.Driver"); } catch
         * (ClassNotFoundException e) { // Could not find the driver class.
         * Likely an issue // with finding the .jar file.
         * System.err.println("Could not find the JDBC driver class.");
         * e.printStackTrace(); return; // Bail out. We cannot do anything
         * further. }
         */
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "vertica");
        myProp.put("loginTimeout", "35");
        myProp.put("binaryBatchInsert", "true");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://V_vmart_node0001.example.com:5433/vmart", myProp);
            System.out.println("Connected!");
            conn.close();
        } catch (SQLTransientConnectionException connException) {
            // There was a potentially temporary network error
            // Could automatically retry a number of times here, but
            // instead just report error and exit.
            System.out.print("Network connection issue: ");
            System.out.println(connException.getMessage());
            return;
        } catch (SQLInvalidAuthorizationSpecException authException) {
            // Either the username or password was wrong
            System.out.println("Could not log into database: ");
            System.out.println(authException.getMessage());
            System.out.println(" Check the login credentials and try again.");
            return;
        } catch (SQLException e) {
            // Catch-all for other exceptions
            e.printStackTrace();
        }
    }
}
```
Usage Considerations

- When you disconnect a user session, any uncommitted transactions are automatically rolled back.

- If your database is not compliant with your Vertica license terms, Vertica issues a SQLWarning when you establish the connection to the database. You can retrieve this warning using the Connection.getWarnings() method. See Managing Licenses in the Administrator's Guide for more information about complying with your license terms.

JDBC Connection Properties

You use connection properties to configure the connection between your JDBC client application and your Vertica database. The properties provide the basic information about the connections, such as the server name and port number to use to connect to your database. They also let you tune the performance of your connection and enable logging.

You can set a connection property in any of three ways:

- Include the property name and value as part of the connection string you pass to the DriverManager.getConnection() method.

- Set the properties in a Properties object, and then pass it to the DriverManager.getConnection() method.

- Use the VerticaConnection.setProperty() method. With this approach, you can change only those connection properties that remain changeable after the connection has been established.

In addition, some of the standard JDBC connection properties have getters and setters on the Connection interface (such as Connection.setAutocommit()).

The following tables list the properties supported by the Vertica JDBC driver, and explain which are required in order for the connection to be established.

Connection Properties

The properties in the following table can only be set before you open the connection to the database. Two of them are required for every connection.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required?</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnSettings</td>
<td>A string containing SQL statements that the JDBC driver automatically runs after it connects to the database. You can use this property to set the locale or schema search path, or perform other configuration that the connection requires.</td>
<td>No</td>
<td>none</td>
</tr>
<tr>
<td>DisableCopyLocal</td>
<td>When set to true, disables file-based COPY LOCAL operations, including copying data from local files and using local files to store data and exceptions. You can use this property to prevent users from writing to and copying from files on a Vertica host, including an MC host.</td>
<td>No</td>
<td>false</td>
</tr>
<tr>
<td>Label</td>
<td>Sets a label for the connection on the server. This value appears in the client_label column of the V_MONITOR.SESSIONS system table.</td>
<td>No</td>
<td>jdbc-driver_version-random_number</td>
</tr>
<tr>
<td>LoginTimeout</td>
<td>The number of seconds Vertica waits for a connection to be established to the database before throwing a SQLException.</td>
<td>No</td>
<td>0 (no TCP timeout)</td>
</tr>
<tr>
<td>SSL</td>
<td>When set to true, use SSL to encrypt the connection to the server. Vertica must be configured to handle SSL connections before you can establish an SSL-encrypted connection to it. See TLS/SSL Server Authentication in the Administrator's Guide.</td>
<td>No</td>
<td>false</td>
</tr>
<tr>
<td>Password</td>
<td>The password to use to log into the database.</td>
<td>Yes</td>
<td>none</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| User                      | The database user name to use to connect to the database. **Required?**: Yes  
**Default Value**: none                                                                                                                                                                                                                                                                                                                                                                                               |
| ConnectionLoadBalance     | A Boolean value indicating whether the client is willing to have its connection redirected to another host in the Vertica database. This setting has an effect only if the server has also enabled connection load balancing. See *About Native Connection Load Balancing* in the Administrator’s Guide for more information about native connection load balancing. **Required?**: No  
**Default Value**: false                                                                                                                                                                                                                                                                                                                                          |
| BackupServerNode          | A string containing the host name or IP address of one or more hosts in the database. If the connection to the host specified in the connection string times out, the client attempts to connect to any host named in this string. The host name or IP address can also include a colon followed by the port number for the database. If no port number is specified, the client uses the standard port number (5433). Separate multiple host name or IP address entries with commas. **Required?**: No  
**Default Value**: none                                                                                                                                                                                                                                                                                                                                         |
| PreferredAddressFamily    | The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:  
- ipv4—Connect to the server using IPv4.  
- ipv6—Connect to the server using IPv6.  
- none—Use the IP address provided by the DNS server. **Required?**: No  
**Default Value**: none                                                                                                                                                                                                                                                                                                                                           |
General Properties

The following properties can be set after the connection is established. None of these properties are required.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCommit</td>
<td>Controls whether the connection automatically commits transactions. Set this parameter to false to prevent the connection from automatically committing its transactions. You often want to do this when you are bulk loading multiple batches of data and you want the ability to roll back all of the loads if an error occurs.</td>
</tr>
<tr>
<td></td>
<td><strong>Previous Property Name</strong>Note:** This property was called defaultAutoCommit in previous versions of the Vertica JDBC driver.</td>
</tr>
<tr>
<td></td>
<td><strong>Set After Connection:</strong> Connection.setAutoCommit()</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> true</td>
</tr>
<tr>
<td>DirectBatchInsert</td>
<td>Determines whether a batch insert stored data directly into ROS (true) or using AUTO mode (false).</td>
</tr>
<tr>
<td></td>
<td>When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.</td>
</tr>
<tr>
<td></td>
<td><strong>Set After Connection:</strong> VerticaConnection.setProperty()</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> false</td>
</tr>
<tr>
<td>MultipleActiveResultSets</td>
<td>Allows more than one active result set on a single connection via MultipleActiveResultSets (MARS).</td>
</tr>
<tr>
<td></td>
<td>If both MultipleActiveResultSets and ResultBufferSize are turned on, MultipleActiveResultSets takes precedence. The connection does not provide an error, however ResultBufferSize is ignored.</td>
</tr>
<tr>
<td></td>
<td><strong>Set After Connection:</strong> VerticaConnection.setProperty()</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> false</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>When set to true, makes the data connection read-only. Any queries attempting to update the database using a read-only connection cause a SQLException. Set After Connection: <code>Connection.setReadOnly()</code>  Default Value: false</td>
</tr>
<tr>
<td>ResultBufferSize</td>
<td>Sets the size of the buffer the Vertica JDBC driver uses to temporarily store result sets. A value of 0 means ResultBufferSize is turned off. Note: This property was named maxLRSMemory in previous versions of the Vertica JDBC driver. Set After Connection: <code>VerticaConnection.setProperty()</code>  Default Value: 8912 (8KB)</td>
</tr>
<tr>
<td>SearchPath</td>
<td>Sets the schema search path for the connection. This value is a string containing a comma-separated list of schema names. See Setting Search Paths in the Administrator’s Guide for more information on the schema search path. Set After Connection: <code>VerticaConnection.setProperty()</code>  Default Value: &quot;$user&quot;, public, v_catalog, v_monitor, v_internal</td>
</tr>
<tr>
<td>ThreePartNaming</td>
<td>A Boolean value that controls how DatabaseMetaData reports the catalog name. When set to true, the database name is returned as the catalog name in the database metadata. When set to false, NULL is returned as the catalog name. Enable this option if your client software is set up to get the catalog name from the database metadata for use in a three-part name reference. Set After Connection: <code>VerticaConnection.setProperty()</code>  Default Value: true</td>
</tr>
<tr>
<td>TransactionIsolation</td>
<td>Sets the isolation level of the transactions that use the</td>
</tr>
</tbody>
</table>
### Logging Properties

The properties that control client logging must be set before the connection is opened. None of these properties are required, and none can be changed after the `Connection` object has been instantiated.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| LogLevel       | Sets the type of information logged by the JDBC driver. The value is set to one of the following values:  
- "DEBUG"  
- "ERROR"  
- "TRACE"  
- "WARNING"  
- "INFO"  
- "OFF"  
  **Default Value:** "OFF" |
| LogNameSpace   | Restricts logging to just messages generated by a specific packages. Valid values are:  
- com.vertica — All messages generated by the JDBC driver  
- com.vertica.jdbc — All messages generated by the top-level |
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC API</td>
<td>All messages generated by the JDBC KV API</td>
</tr>
<tr>
<td>com.vertica.jdbc.core</td>
<td>Connection and statement settings</td>
</tr>
<tr>
<td>com.vertica.jdbc.io</td>
<td>Client/server protocol messages</td>
</tr>
<tr>
<td>com.vertica.jdbc.util</td>
<td>Miscellaneous utilities</td>
</tr>
<tr>
<td>com.vertica.jdbc.dataengine</td>
<td>Query execution and result set iteration</td>
</tr>
<tr>
<td>com.vertica.dataengine</td>
<td>Query execution and result set iteration</td>
</tr>
</tbody>
</table>

**Default Value:** none

**LogPath**

Sets the path where the log file is written.

**Default Value:** The current working directory

### Kerberos Connection Parameters

Use the following parameters to set the service and host name principals for client authentication using Kerberos.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAASConfigName</td>
<td>Provides the name of the JAAS configuration that contains the JAAS Krb5LoginModule and its settings</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> verticajdbc</td>
</tr>
<tr>
<td>KerberosServiceName</td>
<td>Provides the service name portion of the Vertica Kerberos principal, for example: vertica/host@EXAMPLE.COM</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> vertica</td>
</tr>
<tr>
<td>KerberosHostname</td>
<td>Provides the instance or host name portion of the Vertica Kerberos principal, for example: vertica/host@EXAMPLE.COM</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> Value specified in the servername connection string property</td>
</tr>
</tbody>
</table>
Routable Connection API Connection Parameters

Use the following parameters to set properties to enable and configure the connection for Routable Connection lookups.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableRoutableQueries</td>
<td>Enables Routable Connection lookup. See Routing JDBC Queries Directly to a Single Node</td>
</tr>
<tr>
<td></td>
<td>Default Value: false</td>
</tr>
<tr>
<td>FailOnMultiNodePlans</td>
<td>If the query plan requires more than one node, then the query fails. Only applicable when EnableRoutableQueries = true.</td>
</tr>
<tr>
<td></td>
<td>Default Value: true</td>
</tr>
<tr>
<td>MetadataCacheLifetime</td>
<td>The time in seconds to keep projection metadata. Only applicable when EnableRoutableQueries = true.</td>
</tr>
<tr>
<td></td>
<td>Default Value:</td>
</tr>
<tr>
<td>MaxPooledConnections</td>
<td>Cluster-wide maximum number of connections to keep in the VerticaRoutableConnection’s internal pool. Only applicable when EnableRoutableQueries = true.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 20</td>
</tr>
<tr>
<td>MaxPooledConnectionsPerNode</td>
<td>Per-node maximum number of connections to keep in the VerticaRoutableConnection’s internal pool. Only applicable when EnableRoutableQueries = true.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 5</td>
</tr>
</tbody>
</table>

Note: You can also use VerticaConnection.setProperty() method to set properties that have standard JDBC Connection setters, such as AutoCommit.

For information about manipulating these attributes, see Setting and Getting Connection Property Values.

Setting and Getting Connection Property Values

You can set a connection property in any of three ways:
• Include the property name and value as part of the connection string you pass to the 
  DriverManager.getConnection() method.

• Set the properties in a Properties object, and then pass it to the 
  DriverManager.getConnection() method.

• Use the VerticaConnection.setProperty() method. With this approach, you can 
  change only those connection properties that remain changeable after the connection has 
  been established.

In addition, some of the standard JDBC connection properties have getters and setters on the 
Connection interface (such as Connection.setAutocommit()).

Setting Properties When Connecting

There are two ways you can set connection properties when creating a connection to Vertica:

• In the connection string, using the same URL parameter format that you can use to set the 
  username and password. The following example sets the SSL connection parameter to true:

  "jdbc:vertica://VerticaHost:5433/db?user=UserName&password=Password&ssl=true"

Setting a host name using the setProperty() method overrides the host name set in a 
connection string as seen above. If this occurs, Vertica may not be able to connect to a host. 
For example, using the connection string above, the following overrides the VerticaHost 
name:

```java
Properties props = new Properties();
props.setProperty("dataSource", dataSourceURL);
props.setProperty("database", database);
props.setProperty("user", user);
props.setProperty("password", password);
ps.setProperty("jdbcDriver", jdbcDriver);
props.setProperty("hostName", "NonVertica_host);
```

However, if a new connection or override connection is needed, you may enter a valid host 
name in the hostname properties object.

The NonVertica_host hostname overrides VerticaHost name in the connection string. To 
avoid this issue comment out the props.setProperty("hostName", "NonVertica_ 
host"); line:

```java
//props.setProperty("hostName", "NonVertica_host");
```
In a Properties object that you pass to the getConnection() call. You will need to import the java.util.Properties class in order to instantiate a Properties object. Then you use the put() method to add the property name and value to the object:

```java
Properties myProp = new Properties();
myProp.put("user", "ExampleUser");
myProp.put("password", "password123");
myProp.put("LoginTimeout", "35");
Connection conn;
try {
    conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:/ExampleDB", myProp);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Note: The data type of all of the values you set in the Properties object are strings, regardless of the property value's data type.

Getting and Setting Properties After Connecting

The VerticaConnection.getProperty() method lets you get the value of some connection properties. You can use VerticaConnection.setProperty() method to change the value for properties that can be set after the database connection has been established. Since these methods are Vertica-specific, to use them you must cast your Connection object to the VerticaConnection interface. To cast to VerticaConnection, you must either import it into your client application or use a fully-qualified reference (com.vertica.jdbc.VerticaConnection). The following example demonstrates getting and setting the value of the DirectBatchInsert property.

```java
import java.sql.*;
import java.util.Properties;
import com.vertica.jdbc.*;
public class SetConnectionProperties {
    public static void main(String[] args) {
        // Note: If your application needs to run under Java 5, you need to
        // load the JDBC driver using Class.forName() here.
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        // Set DirectBatchInsert to true initially
        myProp.put("DirectBatchInsert", "true");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            // Show state of the DirectBatchInsert property. This was set at the
When run, the example prints the following on the standard output:

DirectBatchInsert state: true
DirectBatchInsert state is now: false

### Setting and Returning a Client Connection Label

The JDBC Client has a method to set and return the client connection label: `getClientInfo()` and `setClientInfo()`. You can use these methods with the SQL Functions `GET_CLIENT_LABEL` and `SET_CLIENT_LABEL`.

When you use these two methods, make sure you pass the string value `APPLICATIONNAME` to both the setter and getter methods.

Use `setClientInfo()` to create a client label, and use `getClientInfo()` to return the client label:
// Could automatically retry a number of times here, but
// instead just report error and exit.
System.out.print("Network connection issue: ");
System.out.print(connException.getMessage());
System.out.println(" Try again later!");
return;
} catch (SQLInvalidAuthorizationSpecException authException) {
    // Either the username or password was wrong
    System.out.print("Could not log into database: ");
    System.out.print(authException.getMessage());
    System.out.println(" Check the login credentials and try again.");
    return;
} catch (SQLException e) {
    // Catch-all for other exceptions
    e.printStackTrace();
}
}

When you run this method, it prints the following result to the standard output:

Connected!
New Conn Label: JDBC Client - Data Load

Setting the Locale for JDBC Sessions

You set the locale for a connection while opening it by including a SET LOCALE statement in the ConnSettings property, or by executing a SET LOCALE statement at any time after opening the connection. Changing the locale of a Connection object affects all of the Statement objects you instantiated using it.

You can get the locale by executing a SHOW LOCALE query. The following example demonstrates setting the locale using ConnSettings and executing a statement, as well as getting the locale:

```java
import java.sql.*;
import java.util.Properties;

public class GetAndSetLocale {
    public static void main(String[] args) {
        // If running under a Java 5 JVM, you need to load the JDBC driver
        // using Class.forName here

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");

        // Set Locale to true en_GB on connection. After the connection
        // is established, the JDBC driver runs the statements in the
        // ConnSettings property.
        myProp.put("ConnSettings", "SET LOCALE TO en_GB");
    }
}
```
Connection conn;
try {
    conn = DriverManager.getConnection(
        "jdbc:vertica://VerticaHost:5433/ExampleDB",
        myProp);

    // Execute a query to get the locale. The results should
    // show "en_GB" as the locale, since it was set by the
    // conn settings property.
    Statement stmt = conn.createStatement();
    ResultSet rs = null;
    rs = stmt.executeQuery("SHOW LOCALE");
    System.out.print("Query reports that Locale is set to: ");
    while (rs.next()) {
        System.out.println(rs.getString(2).trim());
    }

    // Now execute a query to set locale.
    stmt.execute("SET LOCALE TO en_US");

    // Run query again to get locale.
    rs = stmt.executeQuery("SHOW LOCALE");
    System.out.print("Query now reports that Locale is set to: ");
    while (rs.next()) {
        System.out.println(rs.getString(2).trim());
    }

    // Clean up
    conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}

Running the above example displays the following on the system console:

Query reports that Locale is set to: en_GB (LEN)
Query now reports that Locale is set to: en_US (LEN)

Notes:

- JDBC applications use a UTF-16 character set encoding and are responsible for converting
  any non-UTF-16 encoded data to UTF-16. Failing to convert the data can result in errors or
  the data being stored incorrectly.

- The JDBC driver converts UTF-16 data to UTF-8 when passing to the Vertica server and
  converts data sent by Vertica server from UTF-8 to UTF-16.
Changing the Transaction Isolation Level

Changing the transaction isolation level lets you choose how transactions prevent interference from other transactions. By default, the JDBC driver matches the transaction isolation level of the Vertica server. The Vertica default transaction isolation level is READ_COMMITTED, which means any changes made by a transaction cannot be read by any other transaction until after they are committed. This prevents a transaction from reading data inserted by another transaction that is later rolled back.

Vertica also supports the SERIALIZABLE transaction isolation level. This level locks tables to prevent queries from having the results of their WHERE clauses changed by other transactions. Locking tables can have a performance impact, since only one transaction is able to access the table at a time.

A transaction retains its isolation level until it completes, even if the session's isolation level changes during the transaction. Vertica internal processes (such as the Tuple Mover and refresh operations) and DDL operations always run at the SERIALIZABLE isolation level to ensure consistency.

You can change the transaction isolation level connection property after the connection has been established using the Connection object's setter (setTransactionIsolation()) and getter (getTransactionIsolation()). The value for transaction isolation property is an integer. The Connection interface defines constants to help you set the value in a more intuitive manner:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection.TRANSACTION_READ_COMMITTED</td>
<td>2</td>
</tr>
<tr>
<td>Connection.TRANSACTION_SERIALIZABLE</td>
<td>8</td>
</tr>
</tbody>
</table>

**Note:** The Connection interface also defines several other transaction isolation constants (READ_UNCOMMITTED and REPEATABLE_READ). Since Vertica does not support these isolation levels, they are converted to READ_COMMITTED and SERIALIZABLE, respectively.

The following example demonstrates setting the transaction isolation level to SERIALIZABLE.

```java
import java.sql.*;
import java.util.Properties;

public class SetTransactionIsolation {
```
public static void main(String[] args) {
    Properties myProp = new Properties();
    myProp.put("user", "ExampleUser");
    myProp.put("password", "password123");
    Connection conn;
    try {
        conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
        // Get default transaction isolation
        System.out.println("Transaction Isolation Level: "+ conn.getTransactionIsolation());
        // Set transaction isolation to SERIALIZABLE
        conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
        // Get the transaction isolation again
        System.out.println("Transaction Isolation Level: "+ conn.getTransactionIsolation());
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

Running the example results in the following being printed out to the console:

Transaction Isolation Level: 2
Transaction Isolation Level: 8

Using a Pooling Data Source

A pooling data source uses a collection of persistent connections in order to reduce the overhead of repeatedly opening network connections between the client and server. Opening a new connection for each request is costly for both the server and the client than keeping a small pool of connections open constantly, ready to be used by new requests. When a request comes in, one of the pre-existing connections in the pool is assigned to it. Only if there are no free connections in the pool is a new connection created. Once the request is complete, the connection returns to the pool and waits to service another request.

The Vertica JDBC driver supports connection pooling as defined in the JDBC 4.0 standard. If you are using a J2EE-based application server in conjunction with Vertica, it should already have a built-in data pooling feature. All that is required is that the application server work with the PooledConnection interface implemented by Vertica's JDBC driver. An application server's pooling feature is usually well-tuned for the workloads that the server is designed to handle. See your application server's documentation for details on how to work with pooled connections. Normally, using pooled connections should be transparent in your code—you will just open connections and the application server will worry about the details of pooling them.
If you are not using an application server, or your application server does not offer connection pooling that is compatible with Vertica, you can use a third-party pooling library, such as the open-source c3p0 or DBCP libraries, to implement connection pooling.

Note: The Vertica Analytic Database client driver's native connection load balancing feature works with third-party connection pooling supplied by application servers and third-party pooling libraries. See Enabling Native Connection Load Balancing in JDBC for more information.

Enabling Native Connection Load Balancing in JDBC

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen.

If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See About Native Connection Load Balancing in the Administrator's Guide for details.

To enable native load balancing on your client, set the ConnectionLoadBalance connection parameter to true. The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the V_MONITOR.CURRENT_SESSION system table.

```java
import java.sql.*;
import java.util.Properties;
import java.sql.*;
import java.util.Properties;

public class JDBCLoadingBalanceExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "example_password123");
        myProp.put("loginTimeout", "35");
        myProp.put("ConnectionLoadBalance", 1);
        Connection conn;
        for (int x=1; x <= 4; x++) {
            try {
                System.out.print("Connect attempt #" + x + "...");
                conn = DriverManager.getConnection("jdbc:vertica://node01.example.com:5433/vmart", myProp);
```
Statement stmt = conn.createStatement();
// Set the load balance policy to round robin before testing the database's load balancing.
stmt.execute("SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');");
// Query system to table to see what node we are connected to. Assume a single row
// in response set.
ResultSet rs = stmt.executeQuery("SELECT node_name FROM v_monitor.current_session;");
rs.next();
System.out.println("Connected to node " + rs.getString(1).trim());
conn.close();
} catch (SQLTransientConnectionException connException) {
    // There was a potentially temporary network error
    // Could automatically retry a number of times here, but
    // instead just report error and exit.
    System.out.println("Network connection issue: ");
    System.out.println(connException.getMessage());
    System.out.println(" Try again later!");
    return;
} catch (SQLInvalidAuthorizationSpecException authException) {
    // Either the username or password was wrong
    System.out.println("Could not log into database: ");
    System.out.println(authException.getMessage());
    System.out.println(" Check the login credentials and try again.");
    return;
} catch (SQLException e) {
    // Catch-all for other exceptions
    e.printStackTrace();
}
}
}

Running the above example produces the following output:

Connect attempt #1...Connected to node v_vmart_node0002
Connect attempt #2...Connected to node v_vmart_node0003
Connect attempt #3...Connected to node v_vmart_node0001
Connect attempt #4...Connected to node v_vmart_node0002

**JDBC Connection Failover**

If a client application attempts to connect to a host in the Vertica Analytic Database cluster that is down, the connection attempt fails when using the default connection configuration. This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytic Database's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host
specified in the connection settings is unreachable. It gives you two ways to let the client driver automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- Configure your DNS server to return multiple IP addresses for a host name. When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.

- Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.

For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts. Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

Choosing a Failover Method

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

Note: If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytic Database cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytic Database automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytic Database cluster. This may not be possible in your organization.

Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytic Database cluster.
Using DNS Failover

To use DNS failover, you need to change your DNS server’s settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytic Database cluster. You then have all client applications use this host name to connect to Vertica Analytic Database.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.

Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the BackupServerNode parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the BackupServerNode connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```java
import java.sql.*;
import java.util.Properties;

public class ConnectionFailoverExample {
    public static void main(String[] args) {
        // Assume using JDBC 4.0 driver on JVM 6+. No driver loading needed.
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "vertica");
        // Set two backup hosts to be used if connecting to the first host
        // fails. All of these hosts will be tried in order until the connection
        // succeeds or all of the connections fail.
        myProp.put("BackupServerNode", "VerticaHost02,VerticaHost03");
        Connection conn;
        try {
            // The connection string is set to try to connect to a known
            // bad host (in this case, a host that never existed).
            conn = DriverManager.getConnection(
                "jdbc:vertica://BadVerticaHost:5433/vmart", myProp);
            System.out.println("Connected!");
            // Query system to table to see what node we are connected to.
            // Assume a single row in response set.
            Statement stmt = conn.createStatement();
        }
```
ResultSet rs = stmt.executeQuery("SELECT node_name FROM v_monitor.current_session;");
rs.next();
System.out.println("Connected to node " + rs.getString(1).trim());
// Done with connection.
conn.close();
} catch (SQLException e) {
  // Catch-all for other exceptions
  e.printStackTrace();
}
}

When run, the example outputs output similar to the following on the system console:

Connected!
Connected to node v_vmart_node0002

Notice that the connection was made to the first node in the backup list (node 2).

Notes

- When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See Enabling Native Connection Load Balancing in JDBC for more information.
JDBC Data Types

The JDBC driver transparently converts most Vertica data types to the appropriate Java data type. In a few cases, a Vertica data type cannot be directly translated to a Java data type; these exceptions are explained in this section.

The VerticaTypes Class

JDBC does not support all of the data types that Vertica supports. The Vertica JDBC client driver contains an additional class named VerticaTypes that helps you handle identifying these Vertica-specific data types. It contains constants that you can use in your code to specify Vertica data types. This class defines two different categories of data types:

- Vertica's 13 types of interval values. This class contains constant properties for each of these types. You can use these constants to select a specific interval type when instantiating members of the VerticaDayTimeInterval and VerticaYearMonthInterval classes:

```java
// Create a day to second interval.
VerticaDayTimeInterval dayInt = new VerticaDayTimeInterval(VerticaTypes.INTERVAL_DAY_TO_SECOND, 10, 0, 5, 40, 0, 0, false);
// Create a year to month interval.
VerticaYearMonthInterval monthInt = new VerticaYearMonthInterval(VerticaTypes.INTERVAL_YEAR_TO_MONTH, 10, 6, false);
```

- Vertica UUID data type. One way you can use the VerticaTypes.UUID is to query a table's metadata to see if a column is a UUID. See UUID Values for an example.

See the JDBC Documentation for more information on this class.

Numeric Data Alias Conversion

The Vertica server supports data type aliases for integer, float and numeric types. The JDBC driver reports these as its basic data types (BIGINT, DOUBLE PRECISION, and NUMERIC), as follows:

<table>
<thead>
<tr>
<th>Vertica Server Types and Aliases</th>
<th>Vertica JDBC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>BIGINT</td>
</tr>
<tr>
<td>INT</td>
<td></td>
</tr>
</tbody>
</table>
Vertica Server Types and Aliases | Vertica JDBC Type |
---|---|
INT8 |  |
BIGINT |  |
SMALLINT |  |
TINYINT |  |
DOUBLE PRECISION | DOUBLE PRECISION |
FLOAT5 |  |
FLOAT8 |  |
REAL |  |
DECIMAL | NUMERIC |
NUMERIC |  |
NUMBER |  |
MONEY |  |

If a client application retrieves the values into smaller data types, Vertica JDBC driver does not check for overflows. The following example demonstrates the results of this overflow.

```java
import java.sql.*;
import java.util.Properties;

public class JDBC DataTypes {
    public static void main(String[] args) {
        // If running under a Java 5 JVM, use you need to load the JDBC driver using Class.forName here

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/VMart", myProp);
            Statement statement = conn.createStatement();
            // Create a table that will hold a row of different types of numeric data.
            statement.executeUpdate("DROP TABLE IF EXISTS test_all_types cascade");
            statement.executeUpdate("CREATE TABLE test_all_types ("
                    + "c0 INTEGER, c1 TINYINT, c2 DECIMAL, "
                    + "c3 MONEY, c4 DOUBLE PRECISION, c5 REAL)");
            // Add a row of values to it.
            statement.executeUpdate("INSERT INTO test_all_types VALUES("
                    + "111111111111, 444, 55555555555.5555, "
```
The above example prints the following on the console when run:

<table>
<thead>
<tr>
<th>Column 1 (INTEGER)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>getColumnType()</td>
<td>-5</td>
</tr>
<tr>
<td>getColumnTypeName()</td>
<td>BIGINT</td>
</tr>
<tr>
<td>getShort()</td>
<td>455</td>
</tr>
<tr>
<td>getLong()</td>
<td>111111111111</td>
</tr>
<tr>
<td>getInt()</td>
<td>-558038585</td>
</tr>
<tr>
<td>getByte()</td>
<td>-57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column 2 (TINYINT)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>getColumnType()</td>
<td>-5</td>
</tr>
<tr>
<td>getColumnTypeName()</td>
<td>BIGINT</td>
</tr>
<tr>
<td>getShort()</td>
<td>444</td>
</tr>
<tr>
<td>getLong()</td>
<td>444</td>
</tr>
<tr>
<td>getInt()</td>
<td>444</td>
</tr>
<tr>
<td>getByte()</td>
<td>-68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column 3 (DECIMAL)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>getColumnType()</td>
<td>2</td>
</tr>
<tr>
<td>getColumnTypeName()</td>
<td>NUMERIC</td>
</tr>
</tbody>
</table>
Using Intervals with JDBC

The JDBC standard does not contain a data type for intervals (the duration between two points in time). To handle Vertica's INTERVAL data type, you must use JDBC's database-specific object type.

When reading an interval value from a result set, use the ResultSet.getObject() method to retrieve the value, and then cast it to one of the Vertica interval classes: VerticaDayTimeInterval (which represents all ten types of day/time intervals) or VerticaYearMonthInterval (which represents all three types of year/month intervals).

**Note:** The units interval style is not supported. Do not use the SET INTERVALSTYLE statement to change the interval style in your client applications.

Using Intervals in Batch Inserts

When inserting batches into tables that contain interval data, you must create instances of the VerticaDayTimeInterval or VerticaYearMonthInterval classes to hold the data you want to insert. You set values either when calling the class's constructor, or afterwards using setters. You then insert your interval values using the PreparedStatement.setObject() method. You can also use the .setString() method, passing it a string in "DD HH:MM:SS" or "YY-MM" format.
The following example demonstrates inserting data into a table containing a day/time interval and a year/month interval:

```java
import java.sql.*;
import java.util.Properties;
// You need to import the Vertica JDBC classes to be able to instantiate
// the interval classes.
import com.vertica.jdbc.*;

public class IntervalDemo {
    public static void main(String[] args) {
        // If running under a Java 5 JVM, use you need to load the JDBC driver
        // using Class.forName here
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/VMart", myProp);
            // Create table for interval values
            Statement stmt = conn.createStatement();
            stmt.execute("DROP TABLE IF EXISTS interval_demo");
            stmt.executeUpdate("CREATE TABLE interval_demo(" + "DayInt INTERVAL DAY TO SECOND, " + "MonthInt INTERVAL YEAR TO MONTH");
            // Insert data into interval columns using
            // VerticaDayTimeInterval and VerticaYearMonthInterval
            // classes.
            PreparedStatement pstmt = conn.prepareStatement("INSERT INTO interval_demo VALUES(?,?)");
            // Create instances of the Vertica classes that represent
            // intervals.
            VerticaDayTimeInterval dayInt = new VerticaDayTimeInterval(10, 0, 5, 40, 0, 0, false);
            VerticaYearMonthInterval monthInt = new VerticaYearMonthInterval(10, 6, false);
            // These objects can also be manipulated using setters.
            dayInt.setHour(7);
            // Add the interval values to the batch
            ((VerticaPreparedStatement) pstmt).setObject(1, dayInt);
            ((VerticaPreparedStatement) pstmt).setObject(2, monthInt);
            pstmt.addBatch();
            // Set another row from strings.
            // Set day interval in "days HH:MM:SS" format
            pstmt.setString(1, "10 10:10:10");
            // Set year to month value in "MM-YY" format
            pstmt.setString(2, "12-09");
            pstmt.addBatch();
            // Execute the batch to insert the values.
            try {
                pstmt.executeBatch();
            } catch (SQLException e) {
                System.out.println("Error message: " + e.getMessage());
            }
        }
    }
}
```
Reading Interval Values

You read an interval value from a result set using the `ResultSet.getObject()` method, and cast the object to the appropriate Vertica object class: `VerticaDayTimeInterval` for day/time intervals or `VerticaYearMonthInterval` for year/month intervals. This is easy to do if you know that the column contains an interval, and you know what type of interval it is. If your application cannot assume the structure of the data in the result set it reads in, you can test whether a column contains a database-specific object type, and if so, determine whether the object belongs to either the `VerticaDayTimeInterval` or `VerticaYearMonthInterval` classes.

```java
// Retrieve the interval values inserted by previous demo.
// Query the table to get the row back as a result set.
Resultset rs = stmt.executeQuery("SELECT * FROM interval_demo");
// If you do not know the types of data contained in the result set,
// you can read its metadata to determine the type, and use
// additional information to determine the interval type.
ResultSetMetaData md = rs.getMetaData();
for (int x = 1; x <= md.getColumnCount(); x++) {
    // Get data type from metadata
    int colDataType = md.getColumnType(x);
    // You can get the type in a string:
    System.out.println("Column "+x+" is a "
            + md.getColumnTypeName(x));
    // Normally, you'd have a switch statement here to
    // handle all sorts of column types, but this example is
    // simplified to just handle database-specific types
    if (colDataType == Types.OTHER) {
        // Column contains a database-specific type. Determine
        // what type of interval it is. Assuming it is an
        // interval...
        Object columnVal = rs.getObject(x);
        if (columnVal instanceof VerticaDayTimeInterval) {
            // We know it is a date time interval
            VerticaDayTimeInterval interval =
                    (VerticaDayTimeInterval) columnVal;
            // You can use the getters to access the interval's
            // data
            System.out.println("Column " +x+"'s value is ");
            System.out.println(interval.getDay() + " Days ");
            System.out.println(interval.getHour() + " Hours ");
            System.out.println(interval.getMinute() + " Minutes");
        } else if (columnVal instanceof VerticaYearMonthInterval) {
            VerticaYearMonthInterval interval =
                    (VerticaYearMonthInterval) columnVal;
            System.out.println("Column " +x+"'s value is ");
            System.out.println(interval.getYear() + " Years ");
            System.out.println(interval.getMonth() + " Months");
        } else {
            System.out.println("Not an interval.");
        }
    }
}
```
The example prints the following to the console:

Column 1 is a INTERVAL DAY TO SECOND
Column 1’s value is 10 Days 7 Hours 5 Minutes
Column 2 is a INTERVAL YEAR TO MONTH
Column 2’s value is 10 Years 6 Months
Column 1 is a INTERVAL DAY TO SECOND
Column 1’s value is 10 Days 10 Hours 10 Minutes
Column 2 is a INTERVAL YEAR TO MONTH
Column 2’s value is 12 Years 9 Months

Another option is to use database metadata to find columns that contain intervals.

```java
// Determine the interval data types by examining the database // metadata.
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet dbMeta = dbmd.getColumns(null, null, "interval_demo", null);
int colcount = 0;
while (dbMeta.next()) {
    // Get the metadata type for a column.
    int javaType = dbMeta.getInt("DATA_TYPE");
    System.out.println("Column "+ ++colcount + " Type name is " +
                        dbMeta.getString("TYPE_NAME"));

    if(javaType == Types.OTHER) {
        // The SQL_DATETIME_SUB column in the metadata tells you
        // Specifically which subtype of interval you have.
        // The VerticaDayTimeInterval.isDayTimeInterval()
        // methods tells you if that value is a day time.
        //
        int intervalType = dbMeta.getInt("SQL_DATETIME_SUB");
        if(VerticaDayTimeInterval.isDayTimeInterval(intervalType)) {
            // Now you know it is one of the 10 day/time interval types.
            // When you select this column you can cast to
            // VerticaDayTimeInterval.
            // You can get more specific by checking intervalType
            // against each of the 10 constants directly, but
            // they all are represented by the same object.
            System.out.println("column " + colcount + " is a " +
                                "VerticaDayTimeInterval intervalType = "
                                + intervalType);
        } else if(VerticaYearMonthInterval.isYearMonthInterval(  
            intervalType)) {
            //now you know it is one of the 3 year/month intervals,
            //and you can select the column and cast to
            // VerticaYearMonthInterval
            System.out.println("column " + colcount + " is a " +
```


```
"VerticaDayTimeInterval intervalType = "
+ intervalType);
} else {
    System.out.println("Not an interval type.");
}
```

## UUID Values

**UUID** is a core data type in Vertica. However, it is not a core Java data type. You must use the `java.util.UUID` class to represent UUID values in your Java code. The JDBC driver does not translate values from Vertica to non-core Java data types. Therefore, you must send UUID values to Vertica using generic object methods such as `PreparedStatement setObject()`. You also use generic object methods (such as `ResultSet getObject()`) to retrieve UUID values from Vertica. You then cast the retrieved objects as a member of the `java.util.UUID` class.

The following example code demonstrates inserting UUID values into and retrieving UUID values from Vertica.

```java
package jdbc_uuid_example;
import java.sql.*;
import java.util.Properties;
public class VerticaUUIDExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "dbadmin");
        myProp.put("password", "");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://doch01:5433/VMart", myProp);
            Statement stmt = conn.createStatement();

            // Create a table with a UUID column and a VARCHAR column.
            stmt.execute("DROP TABLE IF EXISTS UUID_TEST CASCADE;");
            stmt.execute("CREATE TABLE UUID_TEST (id UUID, description VARCHAR(25));");

            // Prepare a statement to insert a UUID and a string into the table.
            PreparedStatement ps = conn.prepareStatement("INSERT INTO UUID_TEST VALUES (?,?)");
            java.util.UUID uuid; // Holds the UUID value.
            for (Integer x = 0; x < 10; x++) {
                // Generate a random uuid
                uuid = java.util.UUID.randomUUID();
```
// Set the UUID value by calling setObject.
ps setObject(1, uuid);
// Set the String value to indicate which UUID this is.
ps setString(2, "UUID #" + x);
ps.execute();
}

// Query the uuid
ResultSet rs = stmt.executeQuery("SELECT * FROM UUID_TEST ORDER BY description ASC");
while (rs.next()) {
    // Cast the object from the result set as a UUID.
    uuid = (java.util.UUID) rs getObject(1);
    System.out.println(rs getString(2) + " : " + uuid toString());
}
}

The previous example prints output similar to the following:

UUID #0: 67b6dc6-c28c-4965-b9f7-5c8b0a04664d
UUID #1: 485d3835-2887-4233-b003-392254fa97e0
UUID #2: 8142f51-c803-473d-8cfc-c2bc184582a117
UUID #3: bec8b86a-b650-47b0-852c-39229155332d9
UUID #4: 8ae6e3e-c-d43-4ef7-8901-246d0483abf
UUID #5: 669696e-5e86-4e87-8b0-a937f5c187d
UUID #6: 19609ec9-c1-c8-473d-8901-246d0483abf
UUID #7: 97182e0-c-d43-4ef7-8901-246d0483abf
UUID #8: c76c3a2b-a9ef-4d65-b2fb-7c637f872b3c
UUID #9: 3cdbcd26-c177-4277-b3df-bf4389f69d

Determining Whether a Column has a UUID Data Type

JDBC does not support the UUID data type. This limitation means you cannot use the usual ResultSetMetaData.getColumnType() method to determine column's data type is UUID. Calling this method on a UUID column returns Types.OTHER. This value is also to identify interval columns. You can use two ways to determine if a column contains UUIDs:

- Use ResultSetMetaData.getColumnName() to get the name of the column's data type. For UUID columns, this method returns the value "Uuid" as a String.

- Query the table's metadata to get the SQL data type of the column. If this value is equal to VerticaTypes.UUID, the column's data type is UUID.

The following example demonstrates both of these techniques:

// This example assumes you already have a database connection
// and result set from a query on a table that may contain a UUID.
Get the metadata of the result set to get the column definitions
ResultSetMetaData meta = rs.getMetaData();
int colcount;
int maxcol = meta.getColumnCount();
System.out.println("Using column metadata: ");
for (colcount = 1; colcount < maxcol; colcount++) {
    // getColumnType() always returns "OTHER" for UUID columns.
    if (meta.getColumnType(colcount) == Types.OTHER) {
        // To determine that it is a UUID column, test the name of the column type.
        if (meta.getColumnTypeName(colcount).equalsIgnoreCase("uuid")) {
            // It's a UUID column
            System.out.println("Column "+ colcount + " is UUID");
        }
    }
}

// You can also query the table's metadata to find its column types and compare
// it to the VerticaType.UUID constant to see if it is a UUID column.
System.out.println("Using table metadata: ");
DatabaseMetaData dbmd = conn.getMetaData();
// Get the metadata for the previously-created test table.
ResultSet tableMeta = dbmd.getColumns(null, null, "UUID_TEST", null);
colcount = 0;
// Each row in the result set has metadata that describes a single column.
while (tableMeta.next()) {
    colcount++;
    // The SQL_DATA_TYPE column holds the Vertica database data type. You compare
    // this value to the VerticaType.UUID constant to see if it is a UUID.
    if (tableMeta.getInt("SQL_DATA_TYPE") == VerticaTypes.UUID) {
        // Column is a UUID data type...
        System.out.println("Column "+ colcount + " is a UUID column.");
    }
}

This example prints the following to the console if it is run after running the prior example:

Using column metadata:
Column 1 is UUID
Using table metadata:
Column 1 is a UUID column.

Executing Queries Through JDBC

To run a query through JDBC:

2. Run the query.

The method you use to run the query depends on the type of query you want to run:
- a DDL query that does not return a result set.
- a DDL query that returns a result set.
- a DML query

Executing DDL (Data Definition Language) Queries

To run DDL queries, such as `CREATE TABLE` and `COPY`, use the `Statement.execute()` method. You get an instance of this class by calling the `createStatement` method of your connection object.

The following example creates an instance of the `Statement` class and uses it to execute a `CREATE TABLE` and a `COPY` query:

```java
Statement stmt = conn.createStatement();
stmt.execute("CREATE TABLE address_book (Last_Name char(50) default ''," +
    "First_Name char(50), Email char(50), Phone_Number char(50))");
stmt.execute("COPY address_book FROM 'address.dat' DELIMITER ',' NULL 'null'");
```

Executing Queries That Return Result Sets

Use the `Statement` class's `executeQuery` method to execute queries that return a result set, such as `SELECT`. To get the data from the result set, use methods such as `getInt`, `getString`, and `getDouble` to access column values depending upon the data types of columns in the result set. Use `ResultSet.next` to advance to the next row of the data set.

```java
ResultSet rs = null;
rs = stmt.executeQuery("SELECT First_Name, Last_Name FROM address_book");
int x = 1;
while (rs.next()){
    System.out.println(x + ". " + rs.getString(1).trim() + " " + rs.getString(2).trim());
    x++;
}
```

Note: The Vertica JDBC driver does not support scrollable cursors. You can only read forwards through the result set.
Executing DML (Data Manipulation Language) Queries Using executeUpdate

Use the executeUpdate method for DML SQL queries that change data in the database, such as `INSERT`, `UPDATE` and `DELETE` which do not return a result set.

```java
stmt.executeUpdate("INSERT INTO address_book "+
    "VALUES ('Ben-Shachar', 'Tamar', 'tamarrow@example.com'," +
    "'555-380-6466')");
stmt.executeUpdate("INSERT INTO address_book (First_Name, Email) "+
    "VALUES ('Pete','pete@example.com')");
```

**Note:** The Vertica JDBC driver's Statement class supports executing multiple statements in the SQL string you pass to the execute method. The PreparedStatement class does not support using multiple statements in a single execution.
Loading Data Through JDBC

You can use any of the following methods to load data via the JDBC interface:

- Executing a SQL INSERT statement to insert a single row directly.
- Batch loading data using a prepared statement.
- Bulk loading data from files or streams using COPY.

When loading data into Vertica, you need to decide whether to write data to the Write Optimized Store (WOS) or the Read Optimized Store (ROS). By default, most data loading methods insert data into the WOS until it fills up, then insert any additional data directly into ROS containers (called AUTO mode). This is the best method to use when frequently loading small amounts of data (often referred to as trickle-loading). When performing less frequent large data loads (any loads over 100MB of data at once), you should change this behavior to insert data directly into the ROS.

The following sections explain in detail how you load data using JDBC.

Using a Single Row Insert

The simplest way to insert data into a table is to use the SQL INSERT statement. You can use this statement by instantiating a member of the Statement class, and use its executeUpdate() method to run your SQL statement.

The following code fragment demonstrates how you can create a Statement object and use it to insert data into a table named address_book:

```java
Statement stmt = conn.createStatement();
stmt.executeUpdate("INSERT INTO address_book " +
   "VALUES ('Smith', 'John', 'jsmith@example.com', " +
   "'555-123-4567')
```

This method has a few drawbacks: you need convert your data to string and escape any special characters in your data. A better way to insert data is to use prepared statements. See Batch Inserts Using JDBC Prepared Statements.
Using LONG VARCHAR and LONG VARBINARY Data Types with JDBC

Using LONG VARCHAR and LONG VARBINARY data types in a JDBC client application is similar to using VARCHAR and VARBINARY data types. The JDBC driver transparently handles the conversion (for example, between a Java String object and a LONG VARCHAR). The following example code demonstrates inserting and retrieving a LONG VARCHAR string. It uses the JDBC Types class to determine the data type of the string returned by Vertica, although it does not actually need to know whether the database column is a LONG VARCHAR or just a VARCHAR in order to retrieve the value.

```java
import java.sql.*;
import java.util.Properties;

public class LongVarcharExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }

        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            // establish connection and make a table for the data.
            Statement stmt = conn.createStatement();

            // How long we want the example string to be. This is
            // larger than can fit into a traditional VARCHAR (which is limited
            // to 65000.
            int length = 100000;

            // Create a table with a LONG VARCHAR column that can store
            // the string we want to insert.
            stmt.execute("DROP TABLE IF EXISTS longtable CASCADE");
            stmt.execute("CREATE TABLE longtable (text LONG VARCHAR(" + length + "))");

            // Build a long string by appending an integer to a string builder
            // until we hit the size limit. Will result in a string
            // containing 01234567890123....
            StringBuilder sb = new StringBuilder(length);
            for (int i = 0; i < length; i++) {
                sb.append(i % 10);
            }
        }
    }
}
```
String value = sb.toString();
System.out.println("String value is "+value.length()+" characters long.");

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO longtable (text) VALUES(?)");
try {
  // Insert LONG VARCHAR value
  System.out.println("Inserting LONG VARCHAR value");
  pstmt.setString(1, value);
  pstmt.addBatch();
  pstmt.executeBatch();

  // Query the table we created to get the value back.
  ResultSet rs = null;
  rs = stmt.executeQuery("SELECT * FROM longtable");

  // Get metadata about the result set.
  ResultSetMetaData rsmd = rs.getMetaData();
  // Print the type of the first column. Should be
  // LONG VARCHAR. Also check it against the Types class, to
  // recognize it programmatically.
  System.out.println("Column #1 data type is: "+
    rsmd.getColumnTypeName(1));
  if (rsmd.getColumnType(1) == Types.LONGVARCHAR) {
    System.out.println("It is a LONG VARCHAR");
  } else {
    System.out.println("It is NOT a LONG VARCHAR");
  }

  // Print out the string length of the returned value.
  while (rs.next()) {
    // Use the same getString method to get the value that you
    // use to get the value of a VARCHAR.
    System.out.println("Returned string length: "+
      rs.getString(1).length());
  }
} catch (SQLException e) {
  System.out.println("Error message: "+e.getMessage());
  return; // Exit if there was an error
}
// Cleanup
conn.close();
} catch (SQLException e) {
  e.printStackTrace();
}
}

Note: Do not use inefficient encoding formats for LONG VARBINARY and LONG VARCHAR values. Vertica cannot load encoded values larger than 32MB, even if the decoded value is less than 32 MB in size. For example, Vertica returns an error if you attempt to load a 32MB LONG VARBINARY value encoded in octal format, since the
octal encoding quadruples the size of the value (each byte is converted into a backslash followed by three digits).
Batch Inserts Using JDBC Prepared Statements

You can load batches of data into Vertica using prepared `INSERT` statements—server-side statements that you set up once, and then call repeatedly. You instantiate a member of the `PreparedStatement` class with a SQL statement that contains question mark placeholders for data. For example:

```java
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customers(last, first, id) VALUES(?,?,?)");
```

You then set the parameters using data-type-specific methods on the `PreparedStatement` object, such as `setString()` and `setInt()`. Once your parameters are set, call the `addBatch()` method to add the row to the batch. When you have a complete batch of data ready, call the `executeBatch()` method to execute the insert batch.

Behind the scenes, the batch insert is converted into a `COPY` statement. When the connection's AutoCommit parameter is disabled, Vertica keeps the `COPY` statement open and uses it to load subsequent batches until the transaction is committed, the cursor is closed, or your application executes anything else (or executes any statement using another `Statement` or `PreparedStatement` object). Using a single `COPY` statement for multiple batch inserts makes loading data more efficient. If you are loading multiple batches, you should disable the AutoCommit property of the database to take advantage of this increased efficiency.

When performing batch inserts, experiment with various batch and row sizes to determine the settings that provide the best performance.

The following example demonstrates using a prepared statement to batch insert data.

```java
import java.sql.*;
import java.util.Properties;

public class BatchInsertExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");

        // Set streamingBatchInsert to True to enable streaming mode for batch inserts.
        // myProp.put("streamingBatchInsert", "True");

        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            // establish connection and make a table for the data.
            Statement stmt = conn.createStatement();
        }
    }
}
```
// Set AutoCommit to false to allow Vertica to reuse the same
// COPY statement
conn.setAutoCommit(false);

// Drop table and recreate.
stmt.execute("DROP TABLE IF EXISTS customers CASCADE");
stmt.execute("CREATE TABLE customers (CustID int, Last_Name
  + " char(50), First_Name char(50), Email char(50), "
  + " Phone_Number char(12))");

// Some dummy data to insert.
String[] firstNames = new String[] { "Anna", "Bill", "Cindy",
  "Don", "Eric" };
String[] lastNames = new String[] { "Allen", "Brown", "Chu",
  "Dodd", "Estavez" };
String[] emails = new String[] { "aang@example.com",
  "b.brown@example.com", "cindy@example.com",
  "d.d@example.com", "e.estavez@example.com" };
String[] phoneNumbers = new String[] { "123-456-7890",
  "555-444-3333", "555-867-5309",
  "555-555-1212", "781-555-0000" };

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement("" +
  " INSERT INTO customers (CustID, Last_Name, " +
  " First_Name, Email, Phone_Number) " +
  " VALUES(?,?,?,?,?)"");

// Add rows to a batch in a loop. Each iteration adds a
// new row.
for (int i = 0; i < firstNames.length; i++) {
  // Add each parameter to the row.
  pstmt.setInt(1, i + 1);
  pstmt.setString(2, lastNames[i]);
  pstmt.setString(3, firstNames[i]);
  pstmt.setString(4, emails[i]);
  pstmt.setString(5, phoneNumbers[i]);
  // Add row to the batch.
  pstmt.addBatch();
}

try {
  // Batch is ready, execute it to insert the data
  pstmt.executeBatch();
} catch (SQLException e) {
  System.out.println("Error message: " + e.getMessage());
  return; // Exit if there was an error
}

// Commit the transaction to close the COPY command
conn.commit();

// Print the resulting table.
ResultSet rs = null;
rs = stmt.executeQuery("SELECT CustID, First_Name, "
  + " Last_Name FROM customers ORDER BY CustID");
while (rs.next()) {
  System.out.println(rs.getInt(1) + " - "
    + rs.getString(2).trim() + " "
    + rs.getString(3).trim());
}
The result of running the example code is:

1 - Anna Allen
2 - Bill Brown
3 - Cindy Chu
4 - Don Dodd
5 - Eric Estavez

Streaming Batch Inserts

By default, Vertica performs batch inserts by caching each row and inserting the cache when the user calls the `executeBatch()` method. Vertica also supports streaming batch inserts. A streaming batch insert adds a row to the database each time the user calls `addBatch()`. Streaming batch inserts improve database performance by allowing parallel processing and reducing memory demands.

**Note:** Once you begin a streaming batch insert, you cannot make other JDBC calls that require client-server communication until you have executed the batch or closed or rolled back the connection.

To enable streaming batch inserts, set the `streamingBatchInsert` property to True. The preceding code sample includes a line enabling `streamingBatchInsert` mode. Remove the `//` comment marks to enable this line and activate streaming batch inserts.

The following table explains the various batch insert methods and how their behavior differs between default batch insert mode and streaming batch insert mode.

<table>
<thead>
<tr>
<th>Method</th>
<th>Default Batch Insert Behavior</th>
<th>Streaming Batch Insert Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addBatch()</code></td>
<td>Adds a row to the row cache.</td>
<td>Inserts a row into the database.</td>
</tr>
<tr>
<td><code>executeBatch()</code></td>
<td>Adds the contents of the row cache to the database in a single action.</td>
<td>Sends an end-of-batch message to the server and returns an array of integers indicating the success or failure of each <code>addBatch()</code> attempt.</td>
</tr>
</tbody>
</table>
**clearBatch()**  |  Clears the row cache without inserting any rows.  |  Not supported. Triggers an exception if used when streaming batch inserts are enabled.

**Notes**

- Using the `PreparedStatement.setFloat()` method can cause rounding errors. If precision is important, use the `.setDouble()` method instead.

- The `PreparedStatement` object caches the connection's AutoCommit property when the statement is prepared. Later changes to the AutoCommit property have no effect on the prepared statement.

---

**Loading Batches Directly into ROS**

When loading large batches of data (more than 100MB or so), you should load the data directly into ROS containers. Inserting directly into ROS is more efficient for large loads than AUTO mode, since it avoids overflowing the WOS and spilling the remainder of the batch to ROS. Otherwise, the Tuple Mover has to perform a moveout on the data in the WOS, while subsequent data is directly written into ROS containers causing your data to be segmented across storage containers.

When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see [Choosing a Load Method](#).

To directly load batches into ROS, set the `directBatchInsert` connection property to true. See [Setting and Getting Connection Property Values](#) for an explanation of how to set connection properties. When this property is set to true, all batch inserts bypass the WOS and load directly into a ROS container.

If all of batches being inserted using a connection should be inserted into the ROS, you should set the `DirectBatchInsert` connection property to true in the `Properties` object you use to create the connection:

```java
Properties myProp = new Properties();
myProp.put("user", "ExampleUser");
myProp.put("password", "password123");
// Enable directBatchInsert for this connection
myProp.put("DirectBatchInsert", "true");
Connection conn;
try {
    conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
    ...
```
If you will be using the connection for inserting both large and small batches (or you do not know the size batches you will be inserting when you create the Connection object), you can set the DirectBatchInsert property after the connection has been established using the VerticaConnection.setProperty method:

```java
((VerticaConnection)conn).setProperty("DirectBatchInsert", true);
```

See Setting and Getting Connection Property Values for a full example of setting DirectBatchInsert.

**Error Handling During Batch Loads**

When loading individual batches, you can find how many rows were accepted and what rows were rejected (see Identifying Accepted and Rejected Rows for details). If you have disabled the AutoCommit connection setting, other errors (such as disk space errors, for example) do not occur while inserting individual batches. This behavior is caused by having a single SQL COPY statement perform the loading of multiple consecutive batches (which makes the load process more efficient). It is only when the COPY statement closes that the batched data is committed and Vertica reports other types of errors.

Therefore, your bulk loading application should be prepared to check for errors when the COPY statement closes. You can trigger the COPY statement to close by:

- ending the batch load transaction by calling Connection.commit()
- closing the statement using Statement.close()
- setting the connection's AutoCommit property to true before inserting the last batch in the load

**Note:** The COPY statement also closes if you execute any non-insert statement or execute any statement using a different Statement or PreparedStatement object. Ending the COPY statement using either of these methods can lead to confusion and a harder-to-maintain application, since you would need to handle batch load errors in a non-batch load statement. You should explicitly end the COPY statement at the end of your batch load and handle any errors at that time.

**Identifying Accepted and Rejected Rows (JDBC)**

The return value of PreparedStatement.executeQuery is an integer array containing the success or failure status of inserting each row. A value 1 means the row was accepted and a value of -3 means that the row was rejected. In the case where an exception occurred during
the batch execution, you can also get the array using
BatchUpdateException.getUpdateCounts().

The following example extends the example shown in Batch Inserts Using JDBC Prepared
Statements to retrieve this array and display the results the batch load.

```java
import java.sql.*;
import java.util.Arrays;
import java.util.Properties;

public class BatchInsertErrorHandlingExample {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        // establish connection and make a table for the data.
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB",
                    myProp);

            // Disable auto commit
            conn.setAutoCommit(false);

            // Create a statement
            Statement stmt = conn.createStatement();
            // Drop table and recreate.
            stmt.executeUpdate("DROP TABLE IF EXISTS customers CASCADE");
            stmt.executeUpdate("CREATE TABLE customers (CustID int, Last_Name
                    + " char(50), First_Name char(50), Email char(50), 
                    + "Phone_Number char(12))");

            // Some dummy data to insert. The one row won't insert because
            // the phone number is too long for the phone column.
            String[] firstNames = new String[] { "Anna", "Bill", "Cindy", 
                    "Don", "Eric" };
            String[] lastNames = new String[] { "Allen", "Brown", "Chu", 
                    "Dodd", "Estavez" };
            String[] emails = new String[] { "aang@example.com", 
                    "b.brown@example.com", "cindy@example.com", 
                    "d.d@example.com", "e.estavez@example.com" };
            String[] phoneNumbers = new String[] { "123-456-789", 
                    "555-444-3333", "555-867-53093453453", 
                    "555-555-1212", "781-555-0000" };

            // Create the prepared statement
            PreparedStatement pstmt = conn.prepareStatement(
                    "INSERT INTO customers (CustID, Last_Name, " + 
                    "First_Name, Email, Phone_Number) " + 
                    "VALUES(?,?,?,?,?)");

            // Add rows to a batch in a loop. Each iteration adds a
            // new row.
            for (int i = 0; i < firstNames.length; i++) {
                // Add each parameter to the row.
                pstmt.setInt(i, i + 1);
            }
```
pstmt.setString(2, lastNames[i]);
pstmt.setString(3, firstNames[i]);
pstmt.setString(4, emails[i]);
pstmt.setString(5, phoneNumbers[i]);
// Add row to the batch.
pstmt.addBatch();
}

// Integer array to hold the results of inserting
// the batch. Will contain an entry for each row,
// indicating success or failure.
int[] batchResults = null;
try {
    // Batch is ready, execute it to insert the data
    batchResults = pstmt.executeUpdate();
} catch (BatchUpdateException e) {
    // We expect an exception here, since one of the
    // inserted phone numbers is too wide for its column. All of the
    // rest of the rows will be inserted.
    System.out.println("Error message: "+ e.getMessage());

    // Batch results isn't set due to exception, but you
    // can get it from the exception object.
    //
    // In your own code, you shouldn't assume the a batch
    // exception occurred, since exceptions can be thrown
    // by the server for a variety of reasons.
    batchResults = e.getUpdateCounts();
}

// You should also be prepared to catch SQLExceptions in your own
// application code, to handle dropped connections and other general
// problems.

// Commit the transaction
conn.commit();

// Print the array holding the results of the batch insertions.
System.out.println("Return value from inserting batch: "+ Arrays.toString(batchResults));

// Print the resulting table.
ResultSet rs = null;
rs = stmt.executeQuery("SELECT CustID, First_Name, "
    + "Last_Name FROM customers ORDER BY CustID");
while (rs.next()) {
    System.out.println(rs.getInt(1) + " - "
        + rs.getString(2).trim() + " "
        + rs.getString(3).trim());
}

// Cleanup
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}

Running the above example produces the following output on the console:
Error message: [Vertica][VJDBC](100172) One or more rows were rejected by the server. Return value from inserting batch: [1, 1, -3, 1, 1]
1 - Anna Allen
2 - Bill Brown
4 - Don Dodd
5 - Eric Estavez

Notice that the third row failed to insert because its phone number is too long for the Phone_Number column. All of the rest of the rows in the batch (including those after the error) were correctly inserted.

Note: It is more efficient for you to ensure that the data you are inserting is the correct data type and width for the table column you are inserting it into than to handle exceptions after the fact.

Rolling Back Batch Loads on the Server

Batch loads always insert all of their data, even if one or more rows is rejected. Only the rows that caused errors in a batch are not loaded. When the database connection's AutoCommit property is true, batches automatically commit their transactions when they complete, so once the batch finishes loading, the data is committed.

In some cases, you may want all of the data in a batch to be successfully inserted—none of the data should be committed if an error occurs. The best way to accomplish this is to turn off the database connection's AutoCommit property to prevent batches from automatically committing themselves. Then, if a batch encounters an error, you can roll back the transaction after catching the BatchUpdateException caused by the insertion error.

The following example demonstrates performing a rollback if any error occurs when loading a batch.

```java
import java.sql.*;
import java.util.Arrays;
import java.util.Properties;

public class RollbackBatchOnError {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser");
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            // Disable auto-commit. This will allow you to roll back a
            // a batch load if there is an error.
            conn.setAutoCommit(false);
            // establish connection and make a table for the data.
        }
```
Statement stmt = conn.createStatement();
// Drop table and recreate.
stmt.execute("DROP TABLE IF EXISTS customers CASCADE");
stmt.execute("CREATE TABLE customers (CustID int, Last_Name char(50), First_Name char(50), Email char(50), "
+ "Phone_Number char(12))");

// Some dummy data to insert. The one row won't insert because
// the phone number is too long for the phone column.
String[] firstNames = new String[] { "Anna", "Bill", "Cindy", "Don", "Eric" };
String[] emails = new String[] { "aang@example.com", "b.brown@example.com", "cindy@example.com", "d.d@example.com", "e.estavez@example.com" };

// Create the prepared statement
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customers (CustID, Last_Name, "
+ "First_Name, Email, Phone_Number) VALUES(?,?,?,?,
"+ "?))");

// Add rows to a batch in a loop. Each iteration adds a
// new row.
for (int i = 0; i < firstNames.length; i++) {
    // Add each parameter to the row.
    pstmt.setInt(1, i + 1);
    pstmt.setString(2, lastNames[i]);
    pstmt.setString(3, firstNames[i]);
    pstmt.setString(4, emails[i]);
    pstmt.setString(5, phoneNumbers[i]);
    // Add row to the batch.
    pstmt.addBatch();
}

// Integer array to hold the results of inserting
// the batch. Will contain an entry for each row,
// indicating success or failure.
int[] batchResults = null;
try {
    // Batch is ready, execute it to insert the data
    batchResults = pstmt.executeBatch();
    // If we reach here, we inserted the batch without errors.
    // Commit it.
    System.out.println("Batch insert successful. Committing.");
    conn.commit();
} catch (BatchUpdateException e) {
    System.out.println("Error message: " + e.getMessage());
    // Batch results isn't set due to exception, but you
    // can get it from the exception object.
    batchResults = e.getUpdateCounts();
    // Roll back the batch transaction.
    System.out.println("Rolling back batch insertion");
    conn.rollback();
}
}
catch (SQLException e) {
    // General SQL errors, such as connection issues, throw
    // SQLExceptions. Your application should do something more
    // than just print a stack trace,
Running the above example prints the following on the system console:

```
Error message: [Vertica][VJDBC](100172) One or more rows were rejected by the server. Rolling back batch insertion
Return value from inserting batch: [1, 1, -3, 1, 1]
Customers table contains:
```

The return values indicate whether each row was successfully inserted. The value 1 means the row inserted without any issues, and a -3 indicates the row failed to insert.

The customers table is empty since the batch insert was rolled back due to the error caused by the third column.

### Bulk Loading Using the COPY Statement

One of the fastest ways to load large amounts of data into Vertica at once (bulk loading) is to use the COPY statement. This statement loads data from a file stored on a Vertica host (or in a data stream) into a table in the database. You can pass the COPY statement parameters that define the format of the data in the file, how the data is to be transformed as it is loaded, how to handle errors, and how the data should be loaded. See the COPY documentation in the SQL Reference Manual for details.

One parameter that is particularly important is the DIRECT option, which tells COPY to load the data directly into ROS rather than going through the WOS. You should use this option when you are loading large files (over 100MB) into the database. Without this option, your load may
fill the WOS and overflow into ROS, requiring the Tuple Mover to perform a Moveout on the data in the WOS. It is more efficient to directly load into ROS and avoid forcing a moveout.

Only a superuser can use the COPY statement to copy a file stored on a host, so you must connect to the database using a superuser account. If you want to have a non-superuser user bulk-load data, you can use COPY to load from a stream on the host (such as STDIN) rather than a file or stream data from the client (see Streaming Data Via JDBC). You can also perform a standard batch insert using a prepared statement, which uses the COPY statement in the background to load the data.

Note: When using this COPY parameter on any node, confirm that the source file is identical on all nodes. Using different files can produce inconsistent results.

The following example demonstrates using the COPY statement through the JDBC to load a file name customers.txt into a new database table. This file must be stored on the database host to which your application connects (in this example, a host named VerticaHost). Since the customers.txt file used in the example is very large, this example uses the DIRECT option to bypass WOS and load directly into ROS.

```java
import java.sql.*;
import java.util.Properties;
import com.vertica.jdbc.*;

public class COPYFromFile {
    public static void main(String[] args) {
        Properties myProp = new Properties();
        myProp.put("user", "ExampleAdmin"); // Must be superuser
        myProp.put("password", "password123");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB",myProp);
            // Disable AutoCommit
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            // Create a table to hold data.
            stmt.executeUpdate("DROP TABLE IF EXISTS customers;");
            stmt.executeUpdate("CREATE TABLE IF NOT EXISTS customers (Last_Name char(50) "
                + " NOT NULL, First_Name char(50),Email char(50), "
                + " Phone_Number char(15))");

            // Use the COPY command to load data. Load directly into ROS, since
            // this load could be over 100MB. Use ENFORCELENGTH to reject
            // strings too wide for their columns.
            boolean result = stmt.executeUpdate("COPY customers FROM "
                + " '/data/customers.txt' DIRECT ENFORCELENGTH");

            // Determine if execution returned a count value, or a full result
            // set.
            if (result) {
                System.out.println("Got result set");
            } else {
                // Count will usually return the count of rows inserted.
            }
        }
    }
}
```
```java
System.out.println("Got count");
int rowCount = stmt.getUpdateCount();
System.out.println("Number of accepted rows = " + rowCount);

// Commit the data load
conn.commit();
} catch (SQLException e) {
    System.out.print("Error: ");
    System.out.println(e.toString());
    System.out.println(e.toString());
}
```

The example prints the following out to the system console when run (assuming that the customers.txt file contained two million valid rows):

```
Number of accepted rows = 2000000
```

## Streaming Data Via JDBC

There are two options to stream data from a file on the client to your Vertica database:

- Use the VerticaCopyStream class to stream data in an object-oriented manner - details on the class are available in the JDBC Documentation
- Execute a `COPY LOCAL` SQL statement to stream the data

The topics in this section explain how to use these options.

### Using VerticaCopyStream

The VerticaCopyStream class (details on the class are available in the JDBC Documentation) lets you stream data from the client system to a Vertica database. It lets you use the SQL `COPY statement` directly without having to copy the data to a host in the database cluster first. Using the COPY command to load data from the host requires superuser privileges to be able to access the host's filesystem. The COPY statement used to load data from a stream does not require superuser privileges so your client can connect using any user account that has INSERT privileges on the table that will receive the data.

To copy streams into the database:
1. Disable the database connections AutoCommit connection parameter.

2. Instantiate a VerticaCopyStreamObject, passing it at least the database connection objects and a string containing a COPY statement to load the data. This statement must copy data from the STDIN into your table. You can use whatever parameters are appropriate for your data load.

   **Note:** The VerticaCopyStreamObject constructor optionally takes a single InputStream object, or a List of InputStream objects. This option lets you pre-populate the list of streams to be copied into the database.

3. Call VerticaCopyStreamObject.start() to start the COPY statement and begin streaming the data in any streams you have already added to the VerticaCopyStreamObject.

4. Call VerticaCopyStreamObject.addStream() to add additional streams to the list of streams to send to the database. You can then call VerticaCopyStreamObject.execute() to stream them to the server.

5. Optionally, call VerticaCopyStreamObject.getRejects() to get a list of rejected rows from the last .execute() call. The list of rejects is reset by each call to .execute() or .finish().

   **Note:** If you used either the REJECTED DATA or EXCEPTIONS options in the COPY statement you passed to VerticaCopyStreamObject the object in step 2, .getRejects() returns an empty list. You can only use one method of tracking the rejected rows at a time.

6. When you are finished adding streams, call VerticaCopyStreamObject.finish() to send any remaining streams to the database and close the COPY statement.

7. Call Connection.commit() to commit the loaded data.
## Getting Rejected Rows

The `VerticaCopyStreamObject.getRejects()` method returns a List containing the row numbers of rows that were rejected after the previous `.execute()` method call. Each call to `.execute()` clears the list of rejected rows, so you need to call `.getRejects()` after each call to `.execute()`. Since `.start()` and `.finish()` also call `.execute()` to send any pending streams to the server, you should also call `.getRejects()` after these methods as well.

The following example demonstrates loading the content of five text files stored on the client system into a table.

```java
import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;
import com.vertica.jdbc.VerticaConnection;
import com.vertica.jdbc.VerticaCopyStream;

public class CopyMultipleStreamsExample {
    public static void main(String[] args) {
        // Note: If running on Java 5, you need to call Class.forName
        // to manually load the JDBC driver.
        // Set up the properties of the connection
        Properties myProp = new Properties();
        myProp.put("user", "ExampleUser"); // Must be superuser
        myProp.put("password", "password123");
        // When performing bulk loads, you should always disable the
        // connection’s AutoCommit property to ensure the loads happen as
        // efficiently as possible by reusing the same COPY command and
        // transaction.
        myProp.put("AutoCommit", "false");
        Connection conn;
        try {
            conn = DriverManager.getConnection(
                "jdbc:vertica://VerticaHost:5433/ExampleDB", myProp);
            Statement stmt = conn.createStatement();

            // Create a table to receive the data
            stmt.execute("DROP TABLE IF EXISTS customers");
            stmt.execute("CREATE TABLE customers (Last_Name char(50), "
                + "First_Name char(50), Email char(50), "
                + "Phone_Number char(15))");

            // Prepare the query to insert from a stream. This query must use
            // the COPY statement to load data from STDIN. Unlike copying from
            // a file on the host, you do not need superuser privileges to
            // copy a stream. All your user account needs is INSERT privileges
            // on the target table.
            String copyQuery = "COPY customers FROM STDIN "
```
+ "DELIMITER '|' DIRECT ENFORCELENGTH";

// Create an instance of the stream class. Pass in the
// connection and the query string.
VerticaCopyStream stream = new VerticaCopyStream(
    (VerticaConnection) conn, copyQuery);

// Keep running count of the number of rejects
int totalRejects = 0;

// start() starts the stream process, and opens the COPY command.
stream.start();

// If you added streams to VerticaCopyStream before calling start(),
// You should check for rejects here (see below). The start() method
// calls execute() to send any pre-queued streams to the server
// once the COPY statement has been created.

// Simple for loop to load 5 text files named customers-1.txt to
// customers-5.txt
for (int loadNum = 1; loadNum <= 5; loadNum++) {
    // Prepare the input file stream. Read from a local file.
    String filename = "C:\Data\customers-" + loadNum + ".txt";
    System.out.println("\n\nLoading file: " + filename);
    File inputFile = new File(filename);
    FileInputStream inputStream = new FileInputStream(inputStream);

    // Add stream to the VerticaCopyStream
    stream.addStream(inputStream);

    // call execute() to load the newly added stream. You could
    // add many streams and call execute once to load them all.
    // Which method you choose depends mainly on whether you want
    // the ability to check the number of rejections as the load
    // progresses so you can stop if the number of rejects gets too
    // high. Also, high numbers of InputStreams could create a
    // resource issue on your client system.
    stream.execute();

    // Show any rejects from this execution of the stream load
    // getRejects() returns a List containing the
    // row numbers of rejected rows.
    List<Long> rejects = stream.getRejects();

    // The size of the list gives you the number of rejected rows.
    int numRejects = rejects.size();
    totalRejects += numRejects;
    System.out.println("Number of rows rejected in load "+ loadNum + ": " + numRejects);

    // List all of the rows that were rejected.
    Iterator<Long> rejit = rejects.iterator();
    long linecount = 0;
    while (rejit.hasNext()) {
        System.out.print("Rejected row "+ ++linecount);
        System.out.println(" is row " + rejit.next());
    }
}

// Finish closes the COPY command. It returns the number of
// rows inserted.
long results = stream.finish();
System.out.println("Finish returned " + results);

// If you added any streams that hadn't been executed(),
// you should also check for rejects here, since finish()
// calls execute() to

// You can also get the number of rows inserted using
// getRowCount().
System.out.println("Number of rows accepted: 
   + stream.getRowCount());
System.out.println("Total number of rows rejected: " + totalRejects);

// Commit the loaded data
conn.commit();

} catch (Exception e) {
   e.printStackTrace();
}

Running the above example on some sample data results in the following output:

Loading file: C:\Data\customers-1.txt
Number of rows rejected in load #1: 3
Rejected row #1 is row 3
Rejected row #2 is row 7
Rejected row #3 is row 51

Loading file: C:\Data\customers-2.txt
Number of rows rejected in load #2: 5
Rejected row #1 is row 4143
Rejected row #2 is row 6132
Rejected row #3 is row 9998
Rejected row #4 is row 10000
Rejected row #5 is row 10050

Loading file: C:\Data\customers-3.txt
Number of rows rejected in load #3: 9
Rejected row #1 is row 14142
Rejected row #2 is row 16131
Rejected row #3 is row 19999
Rejected row #4 is row 20001
Rejected row #5 is row 20005
Rejected row #6 is row 20049
Rejected row #7 is row 20056
Rejected row #8 is row 20144
Rejected row #9 is row 20236

Loading file: C:\Data\customers-4.txt
Number of rows rejected in load #4: 8
Rejected row #1 is row 23774
Rejected row #2 is row 24141
Rejected row #3 is row 25906
Rejected row #4 is row 26130
Rejected row #5 is row 27317
Rejected row #6 is row 28121
Rejected row #7 is row 29321
Rejected row #8 is row 29998

Loading file: C:\Data\customers-5.txt
Number of rows rejected in load #5: 1
Rejected row #1 is row 39997
Finish returned 39995
Number of rows accepted: 39995
Total number of rows rejected: 26

Note: The above example shows a simple load process that targets one node in the Vertica cluster. It is more efficient to simultaneously load multiple streams to multiple database nodes. Doing so greatly improves performance because it spreads the processing for the load across the cluster.

Using COPY LOCAL with JDBC

To use COPY LOCAL with JDBC, just execute a COPY LOCAL statement with the path to the source file on the client system. This method is simpler than using the VerticaCopyStream class (details on the class are available in the JDBC Documentation). However, you may prefer using VerticaCopyStream if you have many files to copy to the database or if your data comes from a source other than a file (streamed over a network connection, for example).

The following example code demonstrates using COPY LOCAL to copy a file from the client to the database. It is the same as the code shown in Bulk Loading Using the COPY Statement, except for the use of the LOCAL option in the COPY statement, and the path to the data file is on the client system, rather than on the server.

Note: The exceptions/rejections files are created on the client machine when the exceptions and rejected data modifiers are specified on the copy local command. Specify a local path and filename for these modifiers when executing a COPY LOCAL query from the driver.

```java
import java.sql.*;
import java.util.Properties;

public class COPYLocal {
  public static void main(String[] args) {
    // Note: If using Java 5, you must call Class.forName to load the
    // JDBC driver.
    Properties myProp = new Properties();
    myProp.put("user", "ExampleUser"); // Do not need to superuser
    myProp.put("password", "password123");
    Connection conn;
    try {
      conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:5433/ExampleDB",myProp);
      // Disable AutoCommit
      conn.setAutoCommit(false);
      Statement stmt = conn.createStatement();
      // Create a table to hold data.
      stmt.execute("DROP TABLE IF EXISTS customers;");
      stmt.execute("CREATE TABLE IF NOT EXISTS customers (Last_Name char(50) "
```
+ "NOT NULL, First_Name char(50), Email char(50), "
+ "Phone_Number char(15)));

// Use the COPY command to load data. Load directly into ROS, since
// this load could be over 100MB. Use ENFORCELENGTH to reject
// strings too wide for their columns.
boolean result = stmt.executeQuery("COPY customers FROM LOCAL "+"'C:\Data\customers.txt' DIRECT ENFORCELENGTH");

// Determine if execution returned a count value, or a full result
// set.
if (result) {
    System.out.println("Got result set");
} else {
    // Count will usually return the count of rows inserted.
    System.out.println("Got count");
    int rowCount = stmt.getUpdateCount();
    System.out.println("Number of accepted rows = " + rowCount);
}

conn.close();

} catch (SQLException e) {
    System.out.print("Error: ");
    System.out.println(e.toString());
}

The result of running this code appears below. In this case, the customers.txt file contains
10000 rows, seven of which get rejected because they contain data too wide to fit into their
database columns.

Got countNumber of accepted rows = 9993
Handling Errors

When the Vertica JDBC driver encounters an error, it throws a SQLException or one of its subclasses. The specific subclass it throws depends on the type of error that has occurred. Most of the JDBC method calls can result in several different types of errors, in response to which the JDBC driver throws a specific SQLException subclass. Your client application can choose how to react to the error based on the specific exception that the JDBC driver threw.

Note: The specific SQLException subclasses were introduced in the JDBC 4.0 standard. If your client application runs in a Java 5 JVM, it will use the older JDBC 3.0-compliant driver which lacks these subclasses. In that case, all errors throw a SQLException.

The hierarchy of SQLException subclasses is arranged to help your client application determine what actions it can take in response to an error condition. For example:

- The JDBC driver throws SQLTransientException subclasses when the cause of the error may be a temporary condition, such as a timeout error (SQLTimeoutException) or a connection issue (SQLTransientConnectionIssue). Your client application can choose to retry the operation without making any sort of attempt to remedy the error, since it may not reoccur.

- The JDBC driver throws SQLNonTransientException subclasses when the client needs to take some action before it could retry the operation. For example, executing a statement with a SQL syntax error results in the JDBC driver throwing the a SQLSyntaxErrorException (a subclass of SQLNonTransientException). Often, your client application just has to report these errors back to the user and have him or her resolve them. For example, if the user supplied your application with a SQL statement that triggered a SQLSyntaxErrorException, it could prompt the user to fix the SQL error.

See Vertica Analytic Database SQLState Mapping to Java Exception Classes for a list Java exceptions thrown by the JDBC driver.
## Vertica Analytic Database SQLState Mapping to Java Exception Classes

<table>
<thead>
<tr>
<th>SQLSTATE Class or Value</th>
<th>Description</th>
<th>Java Exception Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 00</td>
<td>Successful Completion</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 01</td>
<td>Warning</td>
<td>SQLWarning</td>
</tr>
<tr>
<td>Class 02</td>
<td>No Data</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 03</td>
<td>SQL Statement Not Yet Complete</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 08</td>
<td>Client Connection Exception</td>
<td>SQLNonTransientConnectionException</td>
</tr>
<tr>
<td>Class 09</td>
<td>Triggered Action Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 0A</td>
<td>Feature Not Supported</td>
<td>SQLFeatureNotSupportedException</td>
</tr>
<tr>
<td>Class 0B</td>
<td>Invalid Transaction Initiation</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 0F</td>
<td>Locator Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 0L</td>
<td>Invalid Grantor</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 0P</td>
<td>Invalid Role Specification</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 21</td>
<td>Cardinality Violation</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 22</td>
<td>Data Exception</td>
<td>SQLDataException</td>
</tr>
<tr>
<td>22V21</td>
<td>ERRCODE_INVALID_EPOCH</td>
<td>SQLNonTransientException</td>
</tr>
<tr>
<td>SQLSTATE Class or Value</td>
<td>Description</td>
<td>Java Exception Class</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Class 23</td>
<td>Integrity Constraint Violation</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 24</td>
<td>Invalid Cursor State</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 25</td>
<td>Invalid Transaction State</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 26</td>
<td>Invalid SQL Statement Name</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 27</td>
<td>Triggered Data Change Violation</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 28</td>
<td>Invalid Authorization Specification</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 2B</td>
<td>Dependent Privilege Descriptors Still Exist</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 2D</td>
<td>Invalid Transaction Termination</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 2F</td>
<td>SQL Routine Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 34</td>
<td>Invalid Cursor Name</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 38</td>
<td>External Routine Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 39</td>
<td>External Routine Invocation Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>SQLSTATE Class or Value</td>
<td>Description</td>
<td>Java Exception Class</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Class 3B</td>
<td>Savepoint Exception</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 3D</td>
<td>Invalid Catalog Name</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 3F</td>
<td>Invalid Schema Name</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 40</td>
<td>Transaction Rollback</td>
<td>SQLTransactionRollbackException</td>
</tr>
<tr>
<td>Class 42</td>
<td>Syntax Error or Access Rule Violation</td>
<td>SQLSyntaxErrorException</td>
</tr>
<tr>
<td>Class 44</td>
<td>WITH CHECK OPTION Violation</td>
<td>SQLException</td>
</tr>
<tr>
<td>Class 53</td>
<td>Insufficient Resources</td>
<td>SQLTransientException</td>
</tr>
<tr>
<td>53300</td>
<td>ERRCODE_TOO_MANY_CONNECTIONS</td>
<td>SQLNonTransientConnectionException</td>
</tr>
<tr>
<td>Class 54</td>
<td>Program Limit Exceeded</td>
<td>SQLNonTransientException</td>
</tr>
<tr>
<td>Class 55</td>
<td>Object Not In Prerequisite State</td>
<td>SQLNonTransientException</td>
</tr>
<tr>
<td>55V03</td>
<td>ERRCODE_LOCK_NOT_AVAILABLE</td>
<td>SQLTransactionRollbackException</td>
</tr>
<tr>
<td>Class 57</td>
<td>Operator Intervention</td>
<td>SQLTransientException</td>
</tr>
<tr>
<td>57V01</td>
<td>ERRCODE_ADMIN_SHUTDOWN</td>
<td>SQLNonTransientConnectionException</td>
</tr>
</tbody>
</table>
Routing JDBC Queries Directly to a Single Node

The JDBC driver has the ability to route queries directly to a single node using a special connection called a Routable Connection. This feature is ideal for high-volume "short" requests that return a small number of results that all exist on a single node. The common scenario for using this feature is to do high-volume lookups on data that is identified with a unique key. Routable queries typically provide lower latency and use less system resources than distributed queries. However, the data being queried must be segmented in such a way that the JDBC client can determine on which node the data resides.
Vertica Typical Analytic Query

Typical analytic queries require dense computation on data across all nodes in the cluster and benefit from having all nodes involved in the planning and execution of the queries.

Vertica Routable Query API Query

For high-volume queries that return a single or a few rows of data, it is more efficient to execute the query on the single node that contains the data.

To effectively route a request to a single node, the client must determine the specific node on which the data resides. For the client to be able to determine the correct node, the table must be segmented by one or more columns. For example, if you segment a table on a Primary Key (PK) column, then the client can determine on which node the data resides based on the Primary Key and directly connect to that node to quickly fulfill the request.
The Routable Query API provides two classes for performing routable queries: VerticaRoutableExecutor and VGet. VerticaRoutableExecutor provides a more expressive SQL-based API while VGet provides a more structured API for programmatic access.

- The VerticaRoutableExecutor class allows you to use traditional SQL with a reduced feature set to query data on a single node. Note that the client and server must be at at least Release 7.1 SP1 to use the VerticaRoutableExecutor.

  For joins, the table must be joined on a key column that exists in each table you are joining, and the tables must be segmented on that key. However, this is not true for unsegmented tables, which can always be joined (since all the data in an unsegmented table is available on all nodes).

- The VGet class does not use traditional SQL syntax. Instead, it uses a data structure that you build by defining predicates and predicate expressions and outputs and output expressions. This class is ideal for doing Key/Value type lookups on single tables.

  The data structure used for querying the table must provide a predicate for each segmented column defined in the projection for the table. You must provide, at a minimum, a predicate with a constant value for each segmented column. For example, an `id` with a value of 12234 if the table is segmented only on the `id` column. You can also specify additional predicates for the other, non-segmented, columns in the table. Predicates act like a SQL WHERE clause and multiple predicates/predicate expressions apply together with a SQL AND modifier. Predicates must be defined with a constant value. Predicate expressions can be used to refine the query and can contain any arbitrary SQL expressions (such as less than, greater than, and so on) for any of the non-segmented columns in the table.

Java documentation for all classes and methods in the JDBC Driver is available in the Vertica JDBC Documentation.

Note: The JDBC Routable Query API is read-only and requires JDK 1.6 or greater.

See Also

- Creating Tables and Projections for use with the Routable Query API
- Creating a Connection for Routable Queries
- Defining the Query for Routable Queries Using the VGet Class
Defining the Query for Routable Queries using the VerticaRoutableExecutor Class

Routable Query Performance and Troubleshooting

Creating Tables and Projections for use with the Routable Query API

For routable queries, the client needs to determine the appropriate node to get the data. The client does this by comparing all of the projections available for the table and determining the best projection to use to find the single node that contains data. You must create a projection segmented by the key column(s) on at least one table to take full advantage of the Routable Query API. Other tables which join to this table must either have an unsegmented projection, or a projection segmented as described below.

Note: Tables must be segmented by hash for Routable Queries. See Hash Segmentation Clause. Other segmentation types are not supported.

Creating Tables for use with Routable Queries

To create a table that can be used with the Routable Query API, segment (by hash) the table on a uniformly distributed column. Typically, you segment on a primary key. For faster lookups, sort the projection on the same columns on which you segmented. For example, to create a table that is well suited to Routable Queries:

```sql
CREATE TABLE users ( id INT NOT NULL PRIMARY KEY, username VARCHAR(32), email VARCHAR(64), business_unit VARCHAR(16)) ORDER BY id SEGMENTED BY HASH(id) ALL NODES;
```

This table is segmented based on the `id` column (and ordered by `id` to make lookups faster). To build a query for this table using the Routable Query API, you only need to provide a single predicate for the `id` column which returns a single row when queried.

However, if you were to add multiple columns to the segmentation clause, such as this table:

```sql
CREATE TABLE users2 ( id INT NOT NULL PRIMARY KEY, username VARCHAR(32), email VARCHAR(64), business_unit VARCHAR(16))
```
ORDER BY id, business_unit
SEGMENTED BY HASH(id, business_unit)
ALL NODES;

Then you would need to provide two predicates when querying the *users2* table, since the segmentation clause uses both the *id* and the *business_unit* columns. However, if you know both *id* and *business_unit* when you perform the queries, then it is beneficial to segment on both columns, as it makes it easier for the client to determine that this projection is the best projection to use to determine the correct node.

Designing Tables for Single-node JOINs

If you plan to use the VerticaRoutableExecutor class and join tables during routable queries, then you must segment all tables being joined by the same segmentation key. Typically this key is a Primary/Foreign key on all the tables being joined. For example, the customer_key may be the primary key in a customers dimension table, and the same key is a foreign key in a sales fact table. Projections for a VerticaRoutableExecutor query using these tables must be segmented by hash on the customer key in each table.

If you want to join with small dimension tables, such as date dimensions, then it may be appropriate to make those tables unsegmented so that the *date_dimension* data exists on all nodes. It is important to note that when joining unsegmented tables, you still must specify a segmented table in the createRoutableExecutor() call.

Verifying Existing Projections for Tables

If you have existing tables that are already segmented by hash (for example, on an ID column), then you can determine what predicates are needed to query the table by using the *select get_table_projections('tableName')* command to view the projections associated with the table. The example table displays the following when *select get_table_projections('users')* is run:

```
Projection Name: [Segmented] [Seg Cols] [# of Buddies] [Buddy Projections] [Safe] [UptoDate] [Stats]
----------------------------------------------------------
public.users_b1 [Segmented: Yes] [Seg Cols: "public.users.id"] [K: 1] [public.users_b0] [Safe: Yes] [UptoDate: Yes] [Stats: RowCounts]
public.users_b0 [Segmented: Yes] [Seg Cols: "public.users.id"] [K: 1] [public.users_b1] [Safe: Yes] [UptoDate: Yes] [Stats: RowCounts]
```

Note that for each projection, only the "public.users.id" column is specified, meaning you need to provide a predicate for this column when you build your query.

If the table was segmented on multiple columns, for example *id* and *business_unit*, then you would need to provide both columns as predicates to the routable query.
Creating a Connection for Routable Queries

The JDBC Routable Query API provides the VerticaRoutableConnection (details are available in the JDBC Documentation) interface to connect to a cluster and allow for Routable Queries. This interface provides advanced routing capabilities beyond those of a normal VerticaConnection. The VerticaRoutableConnection provides access to the VerticaRoutableExecutor and VGet classes. See Defining the Query for Routable Queries using the VerticaRoutableExecutor Class and Defining the Query for Routable Queries Using the VGet Class respectively.

You enable access to this class by setting the EnableRoutableQueries JDBC connection property to true.

The VerticaRoutableConnection maintains an internal pool of connections and a cache of table metadata that is shared by all VerticaRoutableExecutor/VGet objects that are produced by the connection's createRoutableExecutor() / prepareGet() method. It is also a fully-fledged JDBC connection on its own and supports all the functionality that a VerticaConnection supports. When this connection is closed, all pooled connections managed by this VerticaRoutableConnection and all child objects are closed too. The connection pool and metadata is only used by child Routable Query operations.

Example:

You can create the connection using a JDBC DataSource:

```java
com.vertica.jdbc.DataSource jdbcSettings = new com.vertica.jdbc.DataSource();
jdbcSettings.setDatabase("exampleDB");
jdbcSettings.setHost("v_vmart_node0001.example.com");
jdbcSettings.setUserID("dbadmin");
jdbcSettings.setPassword("password");
jdbcSettings.setEnableRoutableQueries(true);
jdbcSettings.setPort((short) 5433);

VerticaRoutableConnection conn;
conn = (VerticaRoutableConnection)jdbcSettings.getConnection();
```

You can also create the connection using a connection string and the DriverManager.getConnection() method:

```java
String connectionString = "jdbc:vertica://v_vmart_node0001.example.com:5433/exampleDB?user=dbadmin&password=&EnableRoutableQueries=true";
VerticaRoutableConnection conn = (VerticaRoutableConnection) DriverManager.getConnection(connectionString);
```

Both methods result in a conn connection object that is identical.
Note: Avoid opening many VerticaRoutableConnection connections because this connection maintains its own private pool of connections which are not shared with other connections. Instead, your application should use a single connection and issue multiple queries through that connection.

In addition to the setEnableRoutableQueries property that the Routable Query API adds to the Vertica JDBC connection class, the API also adds additional properties. The complete list is below.

- EnableRoutableQueries: Enables Routable Query lookup capability. Default is false.
- FailOnMultiNodePlans: If the plan requires more than one node, and FailOnMultiNodePlans is true, then the query fails. If it is set to false then a warning is generated and the query continues. However, latency is greatly increased as the Routable Query must first determine the data is on multiple nodes, then a normal query is run using traditional (all node) execution and execution. Defaults to true. Note that this failure cannot occur on simple calls using only predicates and constant values.
- MetadataCacheLifetime: The time in seconds to keep projection metadata. The API caches metadata about the projection used for the query (such as projections). The cache is used on subsequent queries to reduce response time. The default is 300 seconds.
- MaxPooledConnections: Cluster-wide maximum number of connections to keep in the VerticaRoutableConnection’s internal pool. Default 20.
- MaxPooledConnectionsPerNode: Per-node maximum number of connections to keep in the VerticaRoutableConnection’s internal pool. Default 5.

Defining the Query for Routable Queries using the VerticaRoutableExecutor Class

The VerticaRoutableExecutor class is used to access table data directly from a single node. VerticaRoutableExecutor directly queries Vertica only on the node that has all the data needed for the query, avoiding the distributed planning and execution costs associated with a normal Vertica execution. You can use VerticaRoutableExecutor if you need to join tables or use a group by clause, as these operations are not possible using VGet.

When using the VerticaRoutableExecutor class, you must follow these rules:

- If joining tables, all tables being joined must be segmented (by hash) on the same set of columns referenced in the join predicate, unless the table being joined is unsegmented.
When using multiple conditions in WHERE clauses, the WHERE clause must use AND between the conditions. Using OR in the WHERE clause causes the query to degenerate to a multi-node plan. OR, IN list, or range conditions on columns outside of the join condition are acceptable if the data exists on the same node.

You can only execute a single statement per request. "Chained" SQL statements are not permitted.

Your query may be used in a driver-generated subquery to help determine if the query can be executed on a single node. Therefore, you cannot include the semi-colon at the end of the statement and you cannot include SQL comments (using double-dashes, like so: --), as these would cause the driver-generated query to fail.

You create a VerticaRoutableExecutor by calling createRoutableExecutor(schema, table); on a connection object. If schema is set to null, then the search path is used to find the table.

VerticaRoutableExecutor Methods

VerticaRoutableExecutor has the following methods (more details on the class are available in the JDBC Documentation):

- execute(query string, [ column, value | Map ]) - Runs the query. Accepts as input the query to be executed, and either:
  - The column and value when the lookup is being done on just a single value. For example:

    ```java
    String column = "customer_key";
    Integer value = 1;
    ResultSet rs = q.execute(query, column, value)
    ```

  - A Java map of the column names and corresponding values if the lookup is being done on one or more columns. For example: ResultSet rs = q.execute(query, map);. The table must have at least one projection segmented by a set of columns exactly matching the columns in the map. Note that each column defined in the map must have only one value. You cannot include more than one value for the same column. For example:

    ```java
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("customer_key", 1);
    map.put("another_key", 42);
    ResultSet rs = q.execute(query, map);
    ```
The query being executed uses regular SQL. The SQL used must meet the rules of the VerticaRoutableExecutor class. For example, you can add limits and sorts, or use aggregate functions, provided the data exists on a single node.

Important: The JDBC client uses the column/value or map arguments to determine on which node to execute the query. You must make sure that the content of the query uses the same values that you provide in the column/value or map arguments.

Note: The following data types cannot be used as column values. Additionally, if a table is segmented on any columns with the following data types then the table cannot be queried using the Routable Query API:

- interval
- timetz
- timestamptz

Note: The driver does not verify the syntax of the query before it sends the query to the server. If your expression is incorrect, then the query fails.

- close() - Closes this VerticaRoutableExecutor by releasing resources used by this VerticaRoutableExecutor. It does not close the parent JDBC connection to Vertica.
- getWarnings() - Retrieves the first warning reported by calls on this VerticaRoutableExecutor. Additional warnings are chained and can be accessed with the JDBC getNextWarning() method.

Example Query Using VerticaRoutableExecutor

The following example details how to use VerticaRoutableExecutor to execute a query using both a JOIN clause and an aggregate function with a GROUP BY clause. The example also details how to create both a customer and a sales table, and how to segment the tables so they can be joined using the VerticaRoutableExecutor class. This example also uses the date_dimension table from the VMart schema to illustrate how you can also join data on unsegmented tables.

1. Create a table for customer details, and then create the projections which segment on the customer_key.
CREATE TABLE customers (customer_key INT, customer_name VARCHAR(128), customer_email VARCHAR(128));

CREATE PROJECTION cust_proj_b0 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES;

CREATE PROJECTION cust_proj_b1 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 1;

CREATE PROJECTION cust_proj_b2 AS
(SELECT *
 FROM customers) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 2;

SELECT start_refresh();

2. Create a sales table, then create the projections which segment on the customer_key. Since both the customer and sales tables are segmented on the same key, you can join them with the VerticaRoutableExecutor Routable Query lookup.

CREATE TABLE sales (sale_key INT, customer_key INT, date_key INT, sales_amount FLOAT);

CREATE PROJECTION sales_proj_b0 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES;

CREATE PROJECTION sales_proj_b1 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 1;

CREATE PROJECTION sales_proj_b2 AS
(SELECT *
 FROM sales) SEGMENTED BY HASH (customer_key) ALL NODES
OFFSET 2;

SELECT start_refresh();

3. Add some sample data:

INSERT INTO customers VALUES (1, 'Fred', 'fred@example.com');
INSERT INTO customers VALUES (2, 'Sue', 'Sue@example.com');
INSERT INTO customers VALUES (3, 'Dave', 'Dave@example.com');
INSERT INTO customers VALUES (4, 'Ann', 'Ann@example.com');
INSERT INTO customers VALUES (5, 'Jamie', 'Jamie@example.com');
COMMIT;

INSERT INTO sales VALUES(1, 1, 1, '100.00');
INSERT INTO sales VALUES(2, 2, 2, '200.00');
4. Create an unsegmented projection of the VMart date_dimension table for use in this example. Note you must run `SELECT start_refresh();` to unsegment the existing data:

```sql
=> CREATE PROJECTION date_dim_unsegment AS
    (SELECT *
     FROM date_dimension) UNSEGMENTED ALL NODES;

=> SELECT start_refresh();
```

Using the customer, sales, and date_dimension data, you can now create a Routable Query lookup that uses joins and a group by to query the customers table and return the total number of purchases per day for a given customer:

```java
import java.sql.*;
import java.util.HashMap;
import java.util.Map;
import com.vertica.jdbc.kv.*;

public class verticaKV_doc {
    public static void main(String[] args) {
        com.vertica.jdbc.DataSource jdbcSettings = new com.vertica.jdbc.DataSource();
        jdbcSettings.setDatabase("VMart");
        jdbcSettings.setHost("vertica.example.com");
        jdbcSettings.setUserID("dbadmin");
        jdbcSettings.setPassword("password");
        jdbcSettings.setEnableRoutableQueries(true);
        jdbcSettings.setFailOnMultiNodePlans(true);
        jdbcSettings.setPort((short) 5433);
        VerticaRoutableConnection conn;
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("customer_key", 1);
        try {
            conn = (VerticaRoutableConnection) jdbcSettings.getConnection();
            String table = "customers";
            VerticaRoutableExecutor q = conn.createRoutableExecutor(null, table);
            String query = "select d.date, SUM(s.sales_amount) as Total ";
            query += " from customers as c";
            query += " join sales as s ";
            query += " on s.customer_key = c.customer_key ";
            query += " join date_dimension as d ";
            query += " on d.date_key = s.date_key ";
```
```java
query += " where c.customer_key = " + map.get("customer_key");
query += " group by (d.date) order by Total DESC";
ResultSet rs = q.execute(query, map);
while(rs.next()) {
    System.out.print("Date: " + rs.getString("date") + ": ");
    System.out.println("Amount: " + rs.getString("Total"));
}
conn.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

The example code outputs:

```
Date: 2012-01-15: Amount: 900.0
Date: 2012-02-04: Amount: 500.0
Date: 2012-01-01: Amount: 100.0
```

Note that your dates may be different, because the VMart schema randomly generates the dates in the date_dimension table.

### Defining the Query for Routable Queries Using the VGet Class

The VGet class (details on the class are available in the JDBC Documentation) is used to access table data directly from a single node when you do not need to join the data or use a group by clause. Like VerticaRoutableExecutor, VGet directly queries Vertica nodes that have the data needed for the query, avoiding the distributed planning and execution costs associated with a normal Vertica execution. However, VGet does not use SQL. Instead, you define predicates and values to perform Key/Value type lookups on a single table. VGet is especially suited to doing key/value-type lookups on single tables.

You create a VGet by calling `prepareGet(schema, table/proj)` on a connection object. `prepareGet()` takes the name of the schema and the name of a table or projection as arguments.

**VGet Methods**

VGet has the following methods:

- `addPredicate(string, object)` - adds a predicate column and a constant value to the query. You must include a predicate for each column on which the table is segmented. The predicate acts as the "WHERE" clause to the query. Multiple `addPredicate()` method calls are joined by AND modifiers. Note that the VGet retains this value after each call to execute. To remove it, use `ClearPredicates()`.
Note: The following data types cannot be used as predicates. Additionally, if a table is segmented on any columns with the following data types then the table cannot be queried using the Routable Query API:

- interval
- timetz
- timestamptz

- `addPredicateExpression(string)` - Accepts arbitrary SQL expressions that operate on the table's columns as input to the query. Predicate expressions and predicates are joined by AND modifiers. You can use segmented columns in predicate expressions, but they must also be specified as a regular predicate with `addPredicate()`. Note that the VGet retains this value after each call to execute. To remove it, use `ClearPredicates()`.

Note: The driver does not verify the syntax of the expression before it sends it to the server. If your expression is incorrect then the query fails.

- `addOutputColumn(string)` - Adds a column to be included in the output. By default the query runs as `SELECT *` and you do not need to define any output columns to return the data. If you add output columns then you must add all the columns you want returned. Note that the VGet retains this value after each call to execute. To remove it, use `ClearOutputs()`.

- `addOutputExpression(string)` - Accepts arbitrary SQL expressions that operate on the table's columns as output. Note that the VGet retains this value after each call to execute. To remove it, use `ClearOutputs()`.

Note: The driver does not verify the syntax of the expression before it sends it to the server. If your expression is incorrect then the query fails.

Note: `addOutputExpression()` is not supported when querying Flex Tables. If you attempt to use `addOutputExpression()` on a Flex Table query, then a `SQLFeatureNotSupportedException` is thrown.

- `addSortColumn(string, SortOrder)` - Adds a sort order to an output column. The output column can be either the one returned by the default query (SELECT *) or one of the columns defined in `addOutputColumn` or `addOutputExpression`. You can defined multiple sort columns.
setLimit(int) - Sets a limit on the number of results returned. A limit of 0 is unlimited.

clearPredicates() - Removes predicates that were added by addPredicate() and addPredicateExpression().

clearOutputs() - Removes outputs added by addOutput() and addOutputExpression().

clearSortColumns() - Removes sort columns previously added by addSortColumn().

execute() - Runs the query. Care must be taken to ensure that the predicate columns exist on the table and projection used by VGet, and that the expressions do not require multiple nodes to execute. If an expression is sufficiently complex as to require more than one node to execute, execute() throws a SQLException if the FailOnMultiNodePlans connection property is true.

close() - Closes this VGet by releasing resources used by this VGet. It does not close the parent JDBC connection to Vertica.

getWarnings() - Retrieves the first warning reported by calls on this VGet. Additional warnings are chained and can be accessed with the JDBC getNextWarning() method.

You call the execute() method to run query. By default, the VGet fetches all the columns of all the rows that satisfy the logical AND of all the predicates passed via the addPredicate() method. To further customize the get operation use the addOutputColumn(), addOutputExpression(), addPredicateExpression(), addSortColumn() and setLimit() methods.

**Note:** VGet operations span multiple JDBC connections (and multiple Vertica sessions) and do not honor the parent connection's transaction semantics. If consistency is required across multiple executions, the parent VerticaRoutableConnection's consistent read API can be used to guarantee all operations occur at the same epoch.

VGet is thread safe, but all methods are synchronized, so threads that share a VGet instance are never run in parallel. For better parallelism, each thread should have its own VGet instance. Different VGet instances that operate on the same table share pooled connections and metadata in a manner that enables a high degree of parallelism.

**Example**

You can query the table defined in [Creating Tables and Projections for use with the Routable Query API](#) with the following example code. The table defines an id column that is segmented by hash.
import java.sql.*;
import com.vertica.jdbc.kv.*;

public class verticaKV2 {
    public static void main(String[] args) {
        com.vertica.jdbc.DataSource jdbcSettings = new com.vertica.jdbc.DataSource();
        jdbcSettings.setDatabase("exampleDB");
        jdbcSettings.setHost("v_vmart_node0001.example.com");
        jdbcSettings.setUserID("dbadmin");
        jdbcSettings.setPassword("password");
        jdbcSettings.setEnableRoutableQueries(true);
        jdbcSettings.setPort((short) 5433);

        VerticaRoutableConnection conn;
        try {
            conn = (VerticaRoutableConnection) jdbcSettings.getConnection();
            System.out.println("Connected.");
            VGet get = conn.prepareGet("public", "users");
            get.addPredicate("id", 5);
            ResultSet rs = get.execute();
            rs.next();
            System.out.println("ID: " + rs.getString("id"));
            System.out.println("Username: " + rs.getString("username"));
            System.out.println("Email: " + rs.getString("email"));
            System.out.println("Closing Connection.");
            conn.close();
        } catch (SQLException e) {
            System.out.println("Error! Stacktrace:");
            e.printStackTrace();
        }
    }
}

The output:

Connected.
ID: 5
Username: userE
Email: usere@example.com
Closing Connection.

Routing Query Performance and Troubleshooting

This topic details performance considerations and common issues you might encounter when using the Routable Query API.
Using Resource Pools with Routable Queries

Individual Routable Queries are serviced quickly since they directly access a single node and return only one or a few rows of data. However, by default, Vertica resource pools use an AUTO setting for the execution parallelism parameter. When set to AUTO, the setting is determined by the number of CPU cores available and generally results in multi-threaded execution of queries in the resource pool. It is not efficient to create parallel threads on the server because Routable Query operations return data so quickly and Routable Query operations only use a single thread to find a row. To prevent the server from opening unneeded processing threads, you should create a specific resource pool for Routable Query clients. Consider the following settings for the resource pool you use for Routable Queries:

- Set execution parallelism to 1 to force single-threaded queries. This setting improves Routable Query performance.
- Use CPU affinity to limit the resource pool to a specific CPU or CPU set. The setting ensures that the Routable Queries have resources available to them, but it also prevents Routable Queries from significantly impacting performance on the system for other general queries.
- If you do not set a CPU affinity for the resource pool, consider setting the maximum concurrency value of the resource pool to a setting that ensures good performance for Routable Queries, but does not negatively impact the performance of general queries.

Performance Considerations for Routable Query Connections

Because a VerticaRoutableConnection opens an internal pool of connections, it is important to configure MaxPooledConnections and MaxPooledConnectionsPerNode appropriately for your cluster size and the amount of simultaneous client connections. It is possible to impact normal database connections if you are overloading the cluster with VerticaRoutableConnections.

The initial connection to the initiator node discovers all other nodes in the cluster. The internal-pool connections are not opened until a VerticaRoutableExecutor or VGet query is sent. All VerticaRoutableExecutors/VGets in a connection object use connections from the internal pool and are limited by the MaxPooledConnections settings. Connections remain open until they are closed so a new connection can be opened elsewhere if the connection limit has been reached.

Troubleshooting Routable Queries

Routable Query issues generally fall into two categories:
- Not providing enough predicates.
- Queries having to span multiple nodes.

**Predicate Requirements**

You must provide the same number of predicates that correspond to the columns of the table segmented by hash. To determine the segmented columns, run `select get_table_projections('tableName')`. You must provide a predicate for each column displayed in the "Seg Cols" field.

For VGet, this means you must literally use `addPredicate()` to add each of the columns. For VerticaRoutableExecutor, this means you must provide all of the predicates and values in the map sent to `execute()`.

**Multi-node Failures**

It is possible to define the correct number of predicates, but still have a failure because multiple nodes contain the data. This failure occurs because the projection’s data is not segmented in such a way that the data being queried is contained on a single node. Enable logging for the connection and view the logs to verify the projection being used. If the client is not picking the correct projection, then try to query the projection directly by specifying the projection instead of the table in the create/prepare statement, for example:

- Using VerticaRoutableExecutor:
  ```java
  conn.createRoutableExecutor(schema, table/projection);
  ```

- Using VGet:
  ```java
  conn.prepareGet('schema', 'table/projection')
  ```

Additionally, you can use the `EXPLAIN` command in `vsql` to help determine if your query can run in single node. `EXPLAIN` can help you understand why the query is being run as single or multi-node.

**Pre-Segmenting Data Using VHash**

The VHash class is an implementation of the Vertica hash function for use with JDBC client applications.

Hash segmentation in Vertica allows you to segment a projection based on a built-in hash function. The built-in hash function provides even data distribution across some or all nodes in a cluster, resulting in optimal query execution.
Suppose you have several million rows of values spread across thousands of CSV files. Assume that you already have a table segmented by hash. Before you load the values into your database, you probably want to know to which node a particular value loads. For this reason, using VHash can be particularly helpful, by allowing you to pre-segment your data before loading.

The following example shows the VHash class hashing the first column of a file named "testFile.csv". The name of the first column in this file is meterId.

Segment the Data Using VHash

This example demonstrates how you can read the testFile.csv file from the local file system and run a hash function on the meterId column. Using the database metadata from a projection, you can then pre-segment the individual rows in the file based on the hash value of meterId.

```java
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.UnsupportedEncodingException;
import java.util.*;
import java.io.IOException;
import java.sql.*;
import com.vertica.jdbc.kv.VHash;

public class VerticaKVDoc {
    final Map<String, FileOutputStream> files;
    final Map<String, List<Long>> nodeToHashList;
    String segmentationMetadata;
    List<String> lines;

    public static void main(String[] args) throws Exception {
        try {
            Class.forName("com.vertica.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Could not find the JDBC driver class.");
            e.printStackTrace();
            return;
        }

        Properties myProp = new Properties();
        myProp.put("user", "username");
        myProp.put("password", "password");

        VerticaKVDoc ex = new VerticaKVDoc();

        // Read in the data from a CSV file.
        ex.readLinesFromFile("C:\\testFile.csv");

        try (Connection conn = DriverManager.getConnection("jdbc:vertica://VerticaHost:portNumber/databaseName", myProp)) {
```
Compute the hashes and create FileOutputStreams.
ex.prepareForHashing(conn);
}
// Write to files.
ex.writeLinesToFiles();
}

public VerticaKVDoc()
{
    files = new HashMap<String, FileOutputStream>();
    nodeToHashList = new HashMap<String, List<Long>>();
}

public void prepareForHashing(Connection conn) throws SQLException,
FileNotFoundException {
    // Send a query to Vertica to return the projection segments.
    try (ResultSet rs = conn.createStatement().executeQuery(
        "SELECT get_projection_segments('public.projectionName')")
    ) {
        rs.next();
        segmentationMetadata = rs.getString(1);
    }
    // Initialize the data files.
    try (ResultSet rs = conn.createStatement().executeQuery(
        "SELECT node_name FROM nodes")
    ) {
        while (rs.next()) {
            String node = rs.getString(1);
            files.put(node, new FileOutputStream(node + ".csv"));
        }
    }
}

public void writeLinesToFiles() throws UnsupportedEncodingException,
IOException {
    for (String line : lines) {
        long hashedValue = VHash.hashLong(getMeterIdFromLine(line));
        // Write the row data to that node's data file.
        String node = VHash.getNodeFor(segmentationMetadata, hashedValue);
        FileOutputStream fos = files.get(node);
        fos.write(line.getBytes("UTF-8"));
    }
}

private long getMeterIdFromLine(String line) {
    // In our file, "meterId" is the name of the first column in the file.
    return Long.parseLong(line.split(",")[0]);
}

public void readLinesFromFile(String filename) throws IOException {
    lines = new ArrayList<String>();
    String line;
    try (BufferedReader reader = new BufferedReader(
        new FileReader(filename))) {
        while ((line = reader.readLine()) != null) {
            lines.add(line);  
            long hashedValue = VHash.hashLong(getMeterIdFromLine(line));
            // Write the row data to that node's data file.
            String node = VHash.getNodeFor(segmentationMetadata, hashedValue);
            FileOutputStream fos = files.get(node);
            fos.write(line.getBytes("UTF-8"));
        }
    }
}
lines.add(line);

Vertica Analytic Database (9.0.x)
Programming ADO.NET Applications

The Vertica driver for ADO.NET allows applications written in C# to read data from, update, and load data into Vertica databases. It provides a data adapter (Vertica Data Adapter) that facilitates reading data from a database into a data set, and then writing changed data from the data set back to the database. It also provides a data reader (VerticaDataReader) for reading data. The driver requires the .NET framework version 3.5+.

For more information about ADO.NET, see:

- Overview of ADO.NET
- .NET Framework Developer Guide

Note: All of the examples provided in this section are in C#.

ADO.NET Data Types

This table details the mapping between Vertica data types and .NET and ADO.NET data types.

<table>
<thead>
<tr>
<th>.NET Framework DbType</th>
<th>ADO.NET DbType</th>
<th>VerticaType</th>
<th>Vertica Data Type</th>
<th>VerticaDataReader getter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>Bit</td>
<td>Boolean</td>
<td>GetBoolean()</td>
</tr>
<tr>
<td>byte[]</td>
<td>Binary</td>
<td>Binary</td>
<td>Binary</td>
<td>GetBytes()</td>
</tr>
</tbody>
</table>

Note: The limit for LongVarBinary is 32 Million bytes. If you attempt to insert more than the limit during a batch transfer for any one row, then they entire batch fails. Verify the size of the data before...
<table>
<thead>
<tr>
<th>.NET Framework Type</th>
<th>ADO.NET DbType</th>
<th>VerticaType</th>
<th>Vertica Data Type</th>
<th>VerticaDataReader getter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Date Time</td>
<td>Date Time</td>
<td>GetDateTime()</td>
</tr>
<tr>
<td>Datetime</td>
<td>DateTime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Date Time</td>
<td>Date Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time Stamp</td>
<td>Time Stamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>attempting to insert a LongVarBinary during a batch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GetDateTime()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note: The Time portion of the DateTime object for vertica dates is set to DateTime.MinValue. Previously, VerticaType.DateTime was used for all date/time types. VerticaType.DateTime still exists for backwards compatibility, but now there are more specific VerticaTypes for each type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GetDateTimeOffset()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Note: The Date portion of the DateTime is set to DateTime.MinValue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GetDecimal()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GetDouble()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NET Framework Type</td>
<td>ADO.NET DbType</td>
<td>VerticaType</td>
<td>Vertica Data Type</td>
<td>VerticaDataReader getter</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int64</td>
<td>Int64</td>
<td>BigInt</td>
<td>Integer</td>
<td>GetInt64()</td>
</tr>
<tr>
<td>TimeSpan</td>
<td>Object</td>
<td>13 Interval Types</td>
<td>13 Interval Types</td>
<td>GetInterval()</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
<td>Varchar</td>
<td>Varchar</td>
<td>GetString()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LongVarChar</td>
<td>LongVarChar</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>StringFixedLength</td>
<td>Char</td>
<td>Char</td>
<td>GetString()</td>
</tr>
<tr>
<td>Guid</td>
<td>Guid</td>
<td>UUID (see note below)</td>
<td>UUID</td>
<td>GetGuid()</td>
</tr>
<tr>
<td>Object</td>
<td>Object</td>
<td>N/A</td>
<td>N/A</td>
<td>GetValue()</td>
</tr>
</tbody>
</table>

**UUID Backwards Compatibility**

Vertica version 9.0.0 introduced the UUID data type, including JDBC support for UUIDs. The Vertica ADO.NET, ODBC, and OLE DB clients added full support for UUIDs in version 9.0.1.

Note: There are 13 VerticaType values for the 13 types of intervals. The specific VerticaType used determines the conversion rules that the driver applies. Year/Month intervals represented as 365/30 days.
Vertica maintains backwards compatibility with the earlier client driver versions that do not support the UUID data type (see Client Driver and Server Version Compatibility for the versions of the client drivers that Vertica supports). When one of these older clients queries tables with UUID columns in them, Vertica automatically translates the native UUID values to CHAR values. Also, when an older client inserts data into a UUID column, Vertica automatically converts the CHAR value sent by the client into a native UUID value. Vertica also reports the data type of these columns as CHAR when an older client queries a UUID column's metadata.

Setting the Locale for ADO.NET Sessions

- ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions as for ODBC apply if this encoding is violated.

- The ADO.NET driver converts UTF-16 data to UTF-8 when passing to the Vertica server and converts data sent by Vertica server from UTF-8 to UTF-16.

- ADO.NET applications should set the correct server session locale by executing the `SET LOCALE TO` command in order to get expected collation and string functions behavior on the server.

- If there is no default session locale at the database level, ADO.NET applications need to set the correct server session locale by executing the `SET LOCALE TO` command in order to get expected collation and string functions behavior on the server. See the `SET LOCALE` command in the SQL Reference Manual.

Connecting to the Database

This section describes:

- Using SSL: Installing SSL Certificates on Windows
- Opening and Closing the Database Connection (ADO.NET)
- ADO.NET Connection Properties
- Configuring Log Properties
Using SSL: Installing Certificates on Windows

You can optionally secure communication between your ADO.NET application and Vertica using SSL. The Vertica ADO.NET driver uses the default Windows key store when looking for SSL certificates. This is the same key store that Internet Explorer uses.

Before you can use SSL on the client side, you must implement SSL on the server. See TLS/SSL Server Authentication in the Administrator’s Guide, perform those steps, then return to this topic to install the SSL certificate on Windows.

To use SSL for ADO.NET connections to Vertica:

- Import the server and client certificates into the Windows Key Store.
- If required by your certificates, import the public certificate of your Certifying Authority.

Import the Server and Client Certificates into the Windows Key store:

1. Copy the server.crt file you generated when you enabled SSL on the server to your Windows Machine.
2. Double-click the certificate.
3. Let Windows determine the key type, and click Install.

Import the Public Certificate of Your CA:

You must establish a chain of trust for the certificates. You may need to import the public certificate for your Certifying Authority (CA) (especially if it is a self-signed certificate).

1. using the same certificate as above, double-click the certificate.
2. Select Place all certificates in the following store.
3. Click Browse, select Trusted Root Certification Authorities and click Next.
4. Click Install.

Enable SSL in Your ADO.NET Applications

In your connection string, be sure to enable SSL by setting the SSL property in VerticaConnectionStringBuilder to true, for example:
Opening and Closing the Database Connection (ADO.NET)

Before you can access data in Vertica through ADO.NET, you must create a connection to the database using the VerticaConnection class which is an implementation of System.Data.DbConnection. The VerticaConnection class takes a single argument that contains the connection properties as a string. You can manually create a string of property keywords to use as the argument, or you can use the VerticaConnectionStringBuilder class to build a connection string for you.

This topic details the following:

- Manually building a connection string and connecting to Vertica
- Using VerticaConnectionStringBuilder to create the connection string and connecting to Vertica
- Closing the connection

To Manually Create a Connection string:

See ADO.NET Connection Properties for a list of available properties to use in your connection string. At a minimum, you need to specify the Host, Database, and User.

1. For each property, provide a value and append the properties and values one after the other, separated by a semicolon. Assign this string to a variable. For example:

   ```java
   String connectString = "DATABASE=VMart;HOST=v_vmart_node0001;USER=dbadmin";
   ```

2. Build a Vertica connection object that specifies your connection string.

   ```java
   VerticaConnection _conn = new VerticaConnection(connectString)
   ```

3. Open the connection.
4. Create a command object and associate it with a connection. All VerticaCommand objects must be associated with a connection.

```csharp
VerticaCommand command = _conn.CreateCommand();
```

To Use the VerticaConnectionStringBuilder Class to Create a Connection String and Open a connection:

1. Create a new object of the VerticaConnectionStringBuilder class.

```csharp
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
```

2. Update your VerticaConnectionStringBuilder object with property values. See [ADO.NET Connection Properties](#) for a list of available properties to use in your connection string. At a minimum, you need to specify the Host, Database, and User.

```csharp
builder.Host = "v_vmart_node0001";
builder.Database = "VMart";
builder.User = "dbadmin";
```

3. Build a Vertica connection object that specifies your connection

```csharp
VerticaConnection _conn = new VerticaConnection(builder.ToString());
```

4. Open the connection.

```csharp
_conn.Open();
```

5. Create a command object and associate it with a connection. All VerticaCommand objects must be associated with a connection.

```csharp
VerticaCommand command = _conn.CreateCommand();
```

**Note:** If your database is not in compliance with your Vertica license, the call to `VerticaConnection.open()` returns a warning message to the console and the log. See [Managing Licenses](#) in the Administrator's Guide for more information.
To Close the connection:

When you're finished with the database, close the connection. Failure to close the connection can deteriorate the performance and scalability of your application. It can also prevent other clients from obtaining locks.

```csharp
_conn.Close();
```

Example Usage:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            //Perform some operations
            _conn.Close();
        }
    }
}
```

ADO.NET Connection Properties

You use connection properties to configure the connection between your ADO.NET client application and your Vertica database. The properties provide the basic information about the connections, such as the server name and port number, needed to connect to your database.

You can set a connection property in two ways:

- Include the property name and value as part of the connection string you pass to a VerticaConnection.
- Set the properties in a VerticaConnectionStringBuilder object, and then pass the object as a string to a VerticaConnection.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Name of the Vertica database to which you want to connect. For example, if you installed the example VMart database, the database is &quot;VMart&quot;.</td>
<td>none</td>
</tr>
<tr>
<td>User</td>
<td>Name of the user to log into Vertica.</td>
<td>none</td>
</tr>
<tr>
<td>Port</td>
<td>Port on which Vertica is running.</td>
<td>5433</td>
</tr>
<tr>
<td>Host</td>
<td>The host name or IP address of the server on which Vertica is running. You can provide an IPv4 address, IPv6 address, or host name. In mixed IPv4/IPv6 networks, the DNS server configuration determines which IP version address is sent first. Use the PreferredAddressFamily option to force the connection to use either IPv4 or IPv6.</td>
<td>none</td>
</tr>
<tr>
<td>PreferredAddressFamily</td>
<td>The IP version to use if the client and server have both IPv4 and IPv6 addresses and you have provided a host name. Valid values are:</td>
<td>Vertica.Data.VerticaClient.AddressFamilyPreference.None</td>
</tr>
<tr>
<td></td>
<td>- Ipv4—Connect to the server using IPv4.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Ipv6—Connect to the server using IPv6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- None—Use the IP address provided by the DNS</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>server.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>The password associated with the user connecting to the server.</td>
<td>string.Empty</td>
</tr>
<tr>
<td>BinaryTransfer</td>
<td>Provides a Boolean value that, when set to true, uses binary transfer instead of string transfer. When set to false, the ADO.NET connection uses string transfer. Binary transfer provides faster performance in reading data from a server to an ADO.NET client. Binary transfer also requires less bandwidth than string transfer, although it sometimes uses more when transferring a large number of small values.</td>
<td>true</td>
</tr>
<tr>
<td>ConnSettings</td>
<td>SQL commands to run upon connection. Uses %3B for semicolons.</td>
<td>string.Empty</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>Sets the transaction isolation level for Vertica. See Transactions for a description of the different transaction levels. This value is either Serializable, ReadCommitted, or Unspecified. See Setting the Transaction Isolation Level for an example of setting the isolation level using this keyword. Note: By default, this value is set to</td>
<td>System.Data.IsolationLevel.Unspecified</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>IsolationLevel</td>
<td>IsolationLevel.Unspecified, which means the connection uses the server's default transaction isolation level. Vertica's default isolation level is IsolationLevel.ReadCommitted.</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>A string to identify the session on the server.</td>
<td>string</td>
</tr>
<tr>
<td>DirectBatchInsert</td>
<td>A Boolean value, whether to bulk insert to ROS (true) or WOS (false).</td>
<td>false</td>
</tr>
<tr>
<td>ResultBufferSize</td>
<td>The size of the buffer to use when streaming results. A value of 0 means ResultBufferSize is turned off.</td>
<td>8192</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>Number seconds to wait for a connection. A value of 0 means no timeout.</td>
<td>0</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>A Boolean value. If true, throw an exception on write attempts.</td>
<td>false</td>
</tr>
<tr>
<td>Pooling</td>
<td>A boolean value, whether to enable connection pooling. Connection pooling is useful for server applications because it allows the server to reuse connections. This saves resources and enhances the performance of executing commands on the database. It also reduces the amount of time a user must wait to</td>
<td>false</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>establish a connection to the database</td>
<td></td>
</tr>
<tr>
<td>MinPoolSize</td>
<td>An integer that defines the minimum number of connections to pool. <strong>Valid Values:</strong> Cannot be greater than the number of connections that the server is configured to allow. Otherwise, an exception results. <strong>Default:</strong> 55</td>
<td>1</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>An integer that defines the maximum number of connections to pool. <strong>Valid Values:</strong> Cannot be greater than the number of connections that the server is configured to allow. Otherwise, an exception results.</td>
<td>20</td>
</tr>
<tr>
<td>LoadBalanceTimeout</td>
<td>The amount of time, expressed in seconds, to timeout or remove unused pooled connections. <strong>Disable:</strong> Set to 0 (no timeouts) If you are using a cluster environment to load-balance the work, then pool is restricted to the servers in the cluster when the pool was created. If additional servers are added to the cluster, and</td>
<td>0 (no timeout)</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>the pool is not removed, then the new servers are never added to the connection pool unless LoadBalanceTimeout is set and exceeded or VerticaConnection.ClearAllPools() is called manually from an application. If you are using load balancing, then set this property to a value that considers when new servers are added to the cluster. However, do not set it so low that pools are frequently removed and rebuilt, doing so makes pooling ineffective.</td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td>A Boolean value, indicating whether to use SSL for the connection.</td>
<td>false</td>
</tr>
<tr>
<td>IntegratedSecurity</td>
<td>Provides a Boolean value that, when set to true, uses the user’s Windows credentials for authentication, instead of user/password in the connection string.</td>
<td>false</td>
</tr>
<tr>
<td>KerberosServiceName</td>
<td>Provides the service name portion of the Vertica Kerberos principal; for example: vertica/host@EXAMPLE.COM</td>
<td>vertica</td>
</tr>
<tr>
<td>KerberosHostname</td>
<td>Provides the instance or host name portion of the Vertica Kerberos principal; for example:</td>
<td>Value specified in the servername connection string property</td>
</tr>
</tbody>
</table>
Vertica Documentation
Connecting to Vertica

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vertica/host@EXAMPLE.COM</td>
<td></td>
</tr>
</tbody>
</table>

Enabling Native Connection Load Balancing in ADO.NET

Native connection load balancing helps spread the overhead caused by client connections on the hosts in the Vertica database. Both the server and the client must enable native connection load balancing in order for it to have an effect. If both have enabled it, then when the client initially connects to a host in the database, the host picks a host to handle the client connection from a list of the currently up hosts in the database, and informs the client which host it has chosen. If the initially-contacted host did not choose itself to handle the connection, the client disconnects, then opens a second connection to the host selected by the first host. The connection process to this second host proceeds as usual—if SSL is enabled, then SSL negotiations begin, otherwise the client begins the authentication process. See About Native Connection Load Balancing in the Administrator's Guide for details.

To enable native load balancing on your client, set the ConnectionLoadBalance connection parameter to true either in the connection string or using the ConnectionStringBuilder(). The following example demonstrates connecting to the database several times with native connection load balancing enabled, and fetching the name of the node handling the connection from the V_MONITOR.CURRENT_SESSION system table.

```csharp
goto try
```
```csharp
VerticaConnection _conn = new VerticaConnection(builder.ToString());
_conn.Open();
if (i == 1)
{
    // On the first connection, check the server policy for load balance
    VerticaCommand sqlcom = _conn.CreateCommand();
    sqlcom.CommandText = "SELECT LOAD_BALANCE_POLICY FROM V_CATALOG.DATABASES";
    var returnValue = sqlcom.ExecuteScalar();
    Console.WriteLine("Status of load balance policy on server: "+ returnValue.ToString()+"\n");
}
VerticaCommand command = _conn.CreateCommand();
command.CommandText = "SELECT node_name FROM V_MONITOR.CURRENT_SESSION";
VerticaDataReader dr = command.ExecuteReader();
while (dr.Read())
{
    Console.Write("Connect attempt #" + i + "...");
    Console.WriteLine("Connected to node "+ dr[0]);
}
dr.Close();
_conn.Close();
Console.WriteLine("Disconnecting.\n");
```

Running the above example produces the following output:

```
Status of load balance policy on server: roundrobin

Connect attempt #1... Connected to node v_vmart_node0001
Disconnecting.

Connect attempt #2... Connected to node v_vmart_node0002
Disconnecting.

Connect attempt #3... Connected to node v_vmart_node0003
Disconnecting.

Connect attempt #4... Connected to node v_vmart_node0001
Disconnecting.
```

**ADO.NET Connection Failover**

If a client application attempts to connect to a host in the Vertica Analytic Database cluster that is down, the connection attempt fails when using the default connection configuration.
This failure usually returns an error to the user. The user must either wait until the host recovers and retry the connection or manually edit the connection settings to choose another host.

Due to Vertica Analytic Database's distributed architecture, you usually do not care which database host handles a client application's connection. You can use the client driver's connection failover feature to prevent the user from getting connection errors when the host specified in the connection settings is unreachable. It gives you two ways to let the client driver automatically attempt to connect to a different host if the one specified in the connection parameters is unreachable:

- **Configure your DNS server to return multiple IP addresses for a host name.** When you use this host name in the connection settings, the client attempts to connect to the first IP address from the DNS lookup. If the host at that IP address is unreachable, the client tries to connect to the second IP, and so on until it either manages to connect to a host or it runs out of IP addresses.

- **Supply a list of backup hosts for the client driver to try if the primary host you specify in the connection parameters is unreachable.**

For both methods, the process of failover is transparent to the client application (other than specifying the list of backup hosts, if you choose to use the list method of failover). If the primary host is unreachable, the client driver automatically tries to connect to other hosts. Failover only applies to the initial establishment of the client connection. If the connection breaks, the driver does not automatically try to reconnect to another host in the database.

**Choosing a Failover Method**

You usually choose to use one of the two failover methods. However, they do work together. If your DNS server returns multiple IP addresses and you supply a list of backup hosts, the client first tries all of the IPs returned by the DNS server, then the hosts in the backup list.

> **Note:** If a host name in the backup host list resolves to multiple IP addresses, the client does not try all of them. It just tries the first IP address in the list.

The DNS method of failover centralizes the configuration client failover. As you add new nodes to your Vertica Analytic Database cluster, you can choose to add them to the failover list by editing the DNS server settings. All client systems that use the DNS server to connect to Vertica Analytic Database automatically use connection failover without having to change any settings. However, this method does require administrative access to the DNS server that all clients use to connect to the Vertica Analytic Database cluster. This may not be possible in your organization.
Using the backup server list is easier than editing the DNS server settings. However, it decentralizes the failover feature. You may need to update the application settings on each client system if you make changes to your Vertica Analytic Database cluster.

Using DNS Failover

To use DNS failover, you need to change your DNS server’s settings to map a single host name to multiple IP addresses of hosts in your Vertica Analytic Database cluster. You then have all client applications use this host name to connect to Vertica Analytic Database.

You can choose to have your DNS server return as many IP addresses for the host name as you want. In smaller clusters, you may choose to have it return the IP addresses of all of the hosts in your cluster. However, for larger clusters, you should consider choosing a subset of the hosts to return. Otherwise there can be a long delay as the client driver tries unsuccessfully to connect to each host in a database that is down.

Using the Backup Host List

To enable backup list-based connection failover, your client application has to specify at least one IP address or host name of a host in the BackupServerNode parameter. The host name or IP can optionally be followed by a colon and a port number. If not supplied, the driver defaults to the standard Vertica port number (5433). To list multiple hosts, separate them by a comma.

The following example demonstrates setting the BackupServerNode connection parameter to specify additional hosts for the connection attempt. The connection string intentionally has a non-existent node, so that the initial connection fails. The client driver has to resort to trying the backup hosts to establish a connection to Vertica.

```csharp
using System;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder =
                new VerticaConnectionStringBuilder();
            builder.Host = "not.a.real.host:5433";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            builder.BackupServerNode =
                "another.broken.node:5433,v_vmart_node0002.example.com:5433";
            try
```
```csharp
{ 
    VerticaConnection _conn = 
        new VerticaConnection(builder.ToString());
    _conn.Open();
    VerticaCommand sqlcom = _conn.CreateCommand();
    sqlcom.CommandText = "SELECT node_name FROM current_session";
    var returnValue = sqlcom.ExecuteScalar();
    Console.WriteLine("Connected to node: " + 
        returnValue.ToString() + ");"
    _conn.Close();
    Console.WriteLine("Disconnecting.");
} 

catch (Exception e) 
{ 
    Console.WriteLine(e.Message); 
} 
}
```

Notes

- When native connection load balancing is enabled, the additional servers specified in the BackupServerNode connection parameter are only used for the initial connection to a Vertica host. If host redirects the client to another host in the database cluster to handle its connection request, the second connection does not use the backup node list. This is rarely an issue, since native connection load balancing is aware of which nodes are currently up in the database. See [Enabling Native Connection Load Balancing in ADO.NET](#).

- Connections to a host taken from the BackupServerNode list are not pooled for ADO.NET connections.

Configuring Log Properties (ADO.Net)

Log properties for ADO.Net are configured differently than they are other client drivers. On the other client drivers, log properties can be configured as one of the connection properties. The ADO.Net driver user the VerticaLogProperties class to configure the properties.

VerticaLogProperties

VerticaLogProperties is a static class that allows you to set and get the log settings for the ADO.net driver. You can control the log level, log path, and log namespace using this class.

The log is created when the first connection is opened. Once the connection is opened, you cannot change the log path. It must be set prior to opening the connection. You can change the log level and log namespace at any time.
Setting Log Properties

Setting the log properties is done using the three methods in the VerticaLogProperties class. The three methods are:

- `SetLogPath(String path, bool persist)`
- `SetLogNamespace(String lognamespace, bool persist)`
- `SetLogLevel(VerticaLogLevel loglevel, bool persist)`

Each of the methods requires a boolean persist argument. When set to true, the persist argument causes the setting to be written to the client's Windows Registry, where it is used for all subsequent connections. If set to false, then the log property only applies to the current session.

SetLogPath

The SetLogPath method takes as its arguments a string containing the path to the log file and the persist argument. If the path string contains only a directory path, then the log file is created with the name `vdp-driver-MM-dd_HH.mm.ss.log` (where `MM-dd_HH.mm.ss` is the date and time the log was created). If the path ends in a filename, such as `log.txt` or `log.log`, then the log is created with that filename.

If SetLogPath is called with an empty string for the path argument, then the client executable's current directory is used as the log path.

If SetLogPath is not called and no registry entry exists for the log path, and you have called any of the other VerticaLogProperties methods, then the client executable's current directory is used as the log path.

When the persist argument is set to true, the path specified is copied to the registry verbatim. If no filename was specified, then the filename is not saved to the registry.

Note: The path must exist on the client system prior to calling this method. The method does not create directories.

Example Usage:

```
// set the log path
string path = "C:\\log";
VerticaLogProperties.SetLogPath(path, false);
```
SetLogNamespace

The SetLogNamespace method takes as its arguments a string containing the namespace to log and the persist argument. The namespace string to log can be one of the following:

- Vertica
- Vertica.Data.VerticaClient
- Vertica.Data.Internal.IO
- Vertica.Data.Internal.DataEngine
- Vertica.Data.Internal.Core

Namespaces can be truncated to include multiple child namespaces. For example, you can specify "Vertica.Data.Internal" to log for all of the Vertica.Data.Internal namespaces.

If a log namespace is not set, and no value is stored in the registry, then the "Vertica" namespace is used for logging.

Example Usage:

```csharp
// set namespace to log
string lognamespace = "Vertica.Data.VerticaClient";
VerticaLogProperties.SetLogNamespace(lognamespace, false);
```

SetLogLevel

The SetLogLevel method takes as its arguments a VerticaLogLevel type and the persist argument. The VerticaLogLevel argument can be one of:

- VerticaLogLevel.None
- VerticaLogLevel.Fatal
- VerticaLogLevel.Error
- VerticaLogLevel.Warning
- VerticaLogLevel.Info
- VerticaLogLevel.Debug
- VerticaLogLevel.Trace
If a log level is not set, and no value is stored in the registry, then VerticaLogLevel.None is used.

Example Usage:

```csharp
// set log level
VerticaLogLevel level = VerticaLogLevel.Debug;
VerticaLogProperties.SetLogLevel(level, false);
```

Getting Log Properties

You can get the log property values using the getters included in the VerticaLogProperties class. The properties are:

- LogPath
- LogNamespace
- LogLevel

Example Usage:

```csharp
// get current log settings
string logpath = VerticaLogProperties.LogPath;
VerticaLogLevel loglevel = VerticaLogProperties.LogLevel;
string logns = VerticaLogProperties.LogNamespace;
Console.WriteLine("Current Log Settings: ");
Console.WriteLine("Log Path: "+ logpath);
Console.WriteLine("Log Level: "+ loglevel);
Console.WriteLine("Log Namespace: "+ logns);
```

Setting and Getting Log Properties Example

This complete example shows how to set and get log properties:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
```

// configure connection properties
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
builder.Host = "192.168.1.10";
builder.Database = "VMart";
builder.User = "dbadmin";

//get current log settings

string logpath = VerticaLogProperties.LogPath;
VerticaLogLevel loglevel = VerticaLogProperties.LogLevel;
string logns = VerticaLogProperties.LogNamespace;
Console.WriteLine("Old Log Settings:");
Console.WriteLine("Log Path: " + logpath);
Console.WriteLine("Log Level: " + loglevel);
Console.WriteLine("Log Namespace: " + logns);

//set the log path

string path = "C:\\log";
VerticaLogProperties.SetLogPath(path, false);

//set log level

VerticaLogLevel level = VerticaLogLevel.Debug;
VerticaLogProperties.SetLogLevel(level, false);

//set namespace to log

string lognamespace = "Vertica";
VerticaLogProperties.SetLogNamespace(lognamespace, false);

//open the connection

VerticaConnection _conn = new VerticaConnection(builder.ToString());
_conn.Open();

//get new log settings

logpath = VerticaLogProperties.LogPath;
loglevel = VerticaLogProperties.LogLevel;
logns = VerticaLogProperties.LogNamespace;
Console.WriteLine("New Log Settings:");
Console.WriteLine("Log Path: " + logpath);
Console.WriteLine("Log Level: " + loglevel);
Console.WriteLine("Log Namespace: " + logns);

//close the connection

_conn.Close();
}
}

The example produces the following output:
Querying the Database Using ADO.NET

This section describes how to create queries to do the following:

- Inserting data into the database
- Read data from the database
- Load data into the database

**Note:** The `ExecuteNonQuery()` method used to query the database returns an `int32` with the number of rows affected by the query. The maximum size of an `int32` type is a constant and is defined to be $2,147,483,547$. If your query returns more results than the `int32` max, then ADO.NET throws an exception because of the overflow of the `int32` type. However, the query is still processed by Vertica even when the reporting of the return value fails. This is a limitation in .NET, as `ExecuteNonQuery()` is part of the standard ADO.NET interface.

Inserting Data (ADO.NET)

Inserting data can done using the `VerticaCommand` class. `VerticaCommand` is an implementation of `DbCommand`. It allows you to create and send a SQL statement to the database. Use the `CommandText` method to assign a SQL statement to the command and then execute the SQL by calling the `ExecuteNonQuery` method. The `ExecuteNonQuery` method is used for executing statements that do not return result sets.

To Insert a Single Row of data:

1. Create a connection to the database.
2. Create a command object using the connection.
3. Insert data using an INSERT statement. The following is an example of a simple insert. Note that it does not contain a COMMIT statement because the Vertica ADO.NET driver operates in autocommit mode.

```csharp
command.CommandText = "INSERT into test values(2, 'username', 'email', 'password')";
```

4. Execute the query. The rowsAdded variable contains the number of rows added by the insert statement.

```csharp
Int32 rowsAdded = command.ExecuteNonQuery();
```

The ExecuteNonQuery() method returns the number of rows affected by the command for UPDATE, INSERT, and DELETE statements. For all other types of statements it returns -1. If a rollback occurs then it is also set to -1.

**Example Usage:**

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "INSERT into test values(2, 'username', 'email', 'password')";
            Int32 rowsAdded = command.ExecuteNonQuery();
            Console.WriteLine( rowsAdded + " rows added!");
            _conn.Close();
        }
    }
}
```
Using Parameters

You can use parameters to execute similar SQL statements repeatedly and efficiently.
Using Parameters

VerticaParameters are an extension of the System.Data.DbParameter base class in ADO.NET and are used to set parameters in commands sent to the server. Use Parameters in all queries (SELECT/INSERT/UPDATE/DELETE) for which the values in the WHERE clause are not static; that is for all queries that have a known set of columns, but whose filter criteria is set dynamically by an application or end user. Using parameters in this way greatly decreases the chances of a SQL injection issue that can occur when simply creating a SQL query from a number of variables.

Parameters require that a valid DbType, VerticaDbType, or System type be assigned to the parameter. See SQL Data Types and ADO.NET Data Types for a mapping of System, Vertica, and DbType.

To create a parameter placeholder, place either the at sign (@) or a colon (:) character in front of the parameter name in the actual query string. Do not insert any spaces between the placeholder indicator (@ or :) and the placeholder.

**Note:** The @ character is the preferred way to identify parameters. The colon (:) character is supported for backward compatibility.

For example, the following typical query uses the string 'MA' as a filter.

```sql
SELECT customer_name, customer_address, customer_city, customer_state
FROM customer_dimension WHERE customer_state = 'MA';
```

Instead, the query can be written to use a parameter. In the following example, the string MA is replaced by the parameter placeholder @STATE.

```sql
SELECT customer_name, customer_address, customer_city, customer_state
FROM customer_dimension WHERE customer_state = @STATE;
```

For example, the ADO.net code for the prior example would be written as:

```csharp
VerticaCommand command = _conn.CreateCommand();
command.CommandText = "SELECT customer_name, customer_address, customer_city, customer_state
FROM customer_dimension WHERE customer_state = @STATE";
command.Parameters.Add(new VerticaParameter("STATE", VerticaType.VarChar));
command.Parameters["STATE"].Value = "MA";
```

**Note:** Although the VerticaCommand class supports a Prepare() method, you do not need to call the Prepare() method for parameterized statements because Vertica automatically prepares the statement for you.

Creating and Rolling Back Transactions
Creating Transactions

Transactions in Vertica are atomic, consistent, isolated, and durable. When you connect to a database using the Vertica ADO.NET Driver, the connection is in autocommit mode and each individual query is committed upon execution. You can collect multiple statements into a single transaction and commit them at the same time by using a transaction. You can also choose to rollback a transaction before it is committed if your code determines that a transaction should not commit.

Transactions use the VerticaTransaction object, which is an implementation of DbTransaction. You must associate the transaction with the VerticaCommand object.

The following code uses an explicit transaction to insert one row each into tables of the VMart schema.
To Create a Transaction in Vertica Using the ADO.NET driver:

1. Create a connection to the database.

2. Create a command object using the connection.

   ```csharp
   VerticaCommand command = _conn.CreateCommand();
   ```

3. Start an explicit transaction, and associate the command with it.

   ```csharp
   VerticaTransaction txn = _conn.BeginTransaction();
   command.Connection = _conn;
   command.Transaction = txn;
   ```

4. Execute the individual SQL statements to add rows.

   ```csharp
   command.CommandText = "insert into product_dimension values( ... )";
   command.ExecuteNonQuery();
   command.CommandText = "insert into store_orders_fact values( ... )";
   ```

5. Commit the transaction.

   ```csharp
   txn.Commit();
   ```
Rolling Back Transactions

If your code checks for errors, then you can catch the error and rollback the entire transaction.

```csharp
VerticaTransaction txn = _conn.BeginTransaction();
VerticaCommand command = new
    VerticaCommand("insert into product_dimension values( 838929, 5, 'New item 5' )", _conn);
// execute the insert
command.ExecuteNonQuery();
command.CommandText = "insert into product_dimension values( 838929, 6, 'New item 6' )";
// try insert and catch any errors
bool error = false;
try
{
    command.ExecuteNonQuery();
} catch (Exception e)
{
    Console.WriteLine(e.Message);
    error = true;
}
if (error)
{
    txn.Rollback();
    Console.WriteLine("Errors. Rolling Back.");
} else
{
    txn.Commit();
    Console.WriteLine("Queries Successful. Committing.");
}
```
Commit and Rollback Example

This example details how you can commit or rollback queries during a transaction.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VercittaConnectionStringBuilder builder = new VercittaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VercittaConnection _conn = new VercittaConnection(builder.ToString());
            _conn.Open();
            bool error = false;
            VercittaCommand command = _conn.CreateCommand();
            VercittaCommand command2 = _conn.CreateCommand();
            VercittaTransaction txn = _conn.BeginTransaction();
            command.Connection = _conn;
            command.Transaction = txn;
            command.CommandText = "insert into test values(1, 'test', 'test', 'test')";
            Console.WriteLine(command.CommandText);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                error = true;
            }
            command.CommandText = "insert into test values(2, 'ear', 'eye', 'nose', 'extra')";
            Console.WriteLine(command.CommandText);
            try
            {
                command.ExecuteNonQuery();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                error = true;
            }
            if (error)
            {
                txn.Rollback();
                Console.WriteLine("Errors. Rolling Back.");
            }
        }
    }
}```
else
{
    txn.Commit();
    Console.WriteLine("Queries Successful. Committing.");
}
_conn.Close();
}

The example displays the following output on the console:

```
insert into test values(1, 'test', 'test', 'test')
insert into test values(2, 'ear', 'eye', 'nose', 'extra')
[42601]ERROR: INSERT has more expressions than target columns
Errors. Rolling Back.
```

See Also

- Setting the Transaction Isolation Level
Setting the Transaction Isolation Level

You can set the transaction isolation level on a per-connection and per-transaction basis. See Transaction for an overview of the transaction isolation levels supported in Vertica. To set the default transaction isolation level for a connection, use the IsolationLevel keyword in the VerticaConnectionStringBuilder string (see Connection String Keywords for details). To set the isolation level for an individual transaction, pass the isolation level to the VerticaConnection.BeginTransaction() method call to start the transaction.
To set the Isolation Level on a connection-basis:

1. Use the VerticaConnectionStringBuilder to build the connection string.

2. Provide a value for the IsolationLevel builder string. It can take one of two values: IsolationLevel.ReadCommitted (default) or IsolationLevel Serializable. For example:

```csharp
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
builder.Host = "192.168.1.100";
builder.Database = "VMart";
builder.User = "dbadmin";
VerticaConnection _conn1 = new VerticaConnection(builder.ToString());
_conn1.Open();
```
To set the Isolation Level on a Transaction basis:

1. Set the IsolationLevel on the BeginTransaction method, for example

   ```
   VerticaTransaction txn = _conn.BeginTransaction(IsolationLevel.Serializable);
   ```
Example usage:

The following example demonstrates:

- getting the connection's transaction isolation level.
- setting the connection's isolation level using connection property.
- setting the transaction isolation level for a new transaction.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn1 = new VerticaConnection(builder.ToString());
            _conn1.Open();
            VerticaTransaction txn1 = _conn1.BeginTransaction();
            Console.WriteLine("\n Transaction 1 Transaction Isolation Level: " +
                txn1.IsolationLevel.ToString());
            txn1.Rollback();
            VerticaTransaction txn2 = _conn1.BeginTransaction(IsolationLevel.Serializable);
            Console.WriteLine("\n Transaction 2 Transaction Isolation Level: " +
                txn2.IsolationLevel.ToString());
            txn2.Rollback();
            VerticaTransaction txn3 = _conn1.BeginTransaction(IsolationLevel.ReadCommitted);
            Console.WriteLine("\n Transaction 3 Transaction Isolation Level: " +
                txn3.IsolationLevel.ToString());
            _conn1.Close();
        }
    }
}
```

When run, the example code prints the following to the system console:

```
Transaction 1 Transaction Isolation Level: ReadCommitted
Transaction 2 Transaction Isolation Level: Serializable
Transaction 3 Transaction Isolation Level: ReadCommitted
```
Reading Data (ADO.Net)

To read data from the database use VerticaDataReader, an implementation of DbDataReader. This implementation is useful for moving large volumes of data quickly off the server where it can be run through analytic applications.

Note: A VerticaCommand cannot execute anything else while it has an open VerticaDataReader associated with it. To execute something else, close the data reader or use a different VerticaCommand object.

To Read Data From the Database Using VerticaDataReader:

1. Create a connection to the database.
2. Create a command object using the connection.
   ```csharp
   VerticaCommand command = _conn.CreateCommand();
   ```
3. Create a query. This query works with the example VMart database.
   ```sql
   command.CommandText = "SELECT fat_content, product_description " + "FROM (SELECT DISTINCT fat_content, product_description" + " FROM product_dimension " + " WHERE department_description " + " IN ('Dairy') " + " ORDER BY fat_content) AS food " + "LIMIT 10;";
   ```
4. Execute the reader to return the results from the query. The following command calls the ExecuteReader method of the VerticaCommand object to obtain the VerticaDataReader object.
   ```csharp
   VerticaDataReader dr = command.ExecuteReader();
   ```
5. Read the data. The data reader returns results in a sequential stream. Therefore, you must read data from tables row-by-row. The following example uses a while loop to accomplish this:
   ```csharp
   Console.WriteLine("\n\n Fat Content\t Product Description\n------------\t---------------------");
   ```
int rows = 0;
while (dr.Read())
{
    Console.WriteLine("  " + dr[0] + " \t " + dr[1]);
    ++rows;
}
Console.WriteLine("----------\n (" + rows + " rows)\n");

6. When you're finished, close the data reader to free up resources.

dr.Close();

Example Usage:

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "SELECT fat_content, product_description " +
                                    "FROM (SELECT DISTINCT fat_content, product_description" +
                                    " FROM product_dimension " +
                                    " WHERE department_description " +
                                    " IN ('Dairy') " +
                                    " ORDER BY fat_content) AS food " +
                                    "LIMIT 10;";
            VerticaDataReader dr = command.ExecuteReader();
            Console.WriteLine("\n\n Fat Content\t Product Description\n");
            Console.WriteLine("----------\t -----------------------\n");
            int rows = 0;
            while (dr.Read())
            {
                Console.WriteLine("  " + dr[0] + " \t " + dr[1]);
                ++rows;
            }
            Console.WriteLine("----------\n (" + rows + " rows)\n");
            dr.Close();
            _conn.Close();
        }
    }
}
Loading Data Through ADO.Net

This section details the different ways that you can load data in Vertica using the ADO.NET client driver:

- **Using the Vertica Data Adapter**
- **Example Batch Insert Using Parameters and Transactions**
- **Streaming Data Via ADO.NET**

Using the Vertica Data Adapter

The Vertica data adapter (VerticaDataAdapter) enables a client to exchange data between a data set and a Vertica database. It is an implementation of DbDataAdapter. You can use VerticaDataAdapter to simply read data, or, for example, read data from a database into a data set, and then write changed data from the data set back to the database.
**Batching Updates**

When using the `Update()` method to update a dataset, you can optionally use the `UpdateBatchSize()` method prior to calling `Update()` to reduce the number of times the client communicates with the server to perform the update. The default value of `UpdateBatchSize` is 1. If you have multiple `rows.Add()` commands for a data set, then you can change the batch size to an optimal size to speed up the operations your client must perform to complete the update.
Reading Data From Vertica Using the Data adapter:

The following example details how to perform a select query on the VMart schema and load the result into a DataTable, then output the contents of the DataTable to the console.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            // Try/Catch any exceptions
            try
            {
                using (_conn)
                {
                    // Create the command
                    VerticaCommand command = _conn.CreateCommand();
                    command.CommandText = "select product_key, product_description " +
                        "from product_dimension where product_key < 10";

                    // Associate the command with the connection
                    command.Connection = _conn;

                    // Create the DataAdapter
                    VerticaDataAdapter adapter = new VerticaDataAdapter();
                    adapter.SelectCommand = command;

                    // Fill the DataTable
                    DataTable table = new DataTable();
                    adapter.Fill(table);

                    // Display each row and column value.
                    int i = 1;
                    foreach (DataRow row in table.Rows)
                    {
                        foreach (DataColumn column in table.Columns)
                        {
                            Console.Write(row[column] + "\t");
                        }
                    }
                }
            }
        }
    }
}
```csharp
    } else {
        Console.WriteLine(i + " rows returned.");
    }
    Console.WriteLine();
    i++;

    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    _conn.Close();
}
```
Reading Data From Vertica into a Data set and Changing data:

The following example shows how to use a data adapter to read from and insert into a dimension table of the VMart schema.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Vertica.Data.VerticaClient
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            // Try/Catch any exceptions
            try
            {
                using (_conn)
                {
                    //Create a data adapter object using the connection
                    VerticaDataAdapter da = new VerticaDataAdapter();

                    //Create a select statement that retrieves data from the table
                    VerticaCommand da.SelectCommand = new
                    VerticaCommand("select * from product_dimension where product_key < 10",
                    _conn);
                    //Set up the insert command for the data adapter, and bind variables for
                    some of the columns
                    da.InsertCommand = new
                    VerticaCommand("insert into product_dimension values( :key, :version, :desc 
                    )",
                    _conn);
                    da.InsertCommand.Parameters.Add(new VerticaParameter("key", VerticaType.BigInt));
                    VerticaType.BigInt));
                    VerticaType.VarChar));
                    da.InsertCommand.Parameters[0].SourceColumn = "product_key";
                    da.InsertCommand.Parameters[1].SourceColumn = "product_version";
                    da.TableMappings.Add("product_key", "product_key");
                }
            }
        }
    }
}
```
da.TableMappings.Add("product_version", "product_version");
da.TableMappings.Add("product_description", "product_description");

//Create and fill a Data set for this dimension table, and get the resulting DataTable.
DataSet ds = new DataSet();
da.Fill(ds, 0, 0, "product_dimension");
DataTable dt = ds.Tables[0];

//Bind parameters and add two rows to the table.
DataRow dr = dt.NewRow();
  dr["product_key"] = 838929;
  dr["product_version"] = 5;
  dr["product_description"] = "New item 5";
  dt.Rows.Add(dr);
  dr = dt.NewRow();
  dr["product_key"] = 838929;
  dr["product_version"] = 6;
  dr["product_description"] = "New item 6";
  dt.Rows.Add(dr);

//Extract the changes for the added rows.
DataSet ds2 = ds.GetChanges();

//Send the modifications to the server.
int updateCount = da.Update(ds2, "product_dimension");

//Merge the changes into the original Data set, and mark it up to date.
ds.Merge(ds2);
ds.AcceptChanges();
Console.WriteLine(updateCount + " updates made!");
}

catch (Exception e)
{
    Console.WriteLine(e.Message);
}
_conn.Close();
}
Example Batch Insert Using Parameters and Transactions

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            // Create arrays for column data
            int[] ids = {1, 2, 3, 4};
            string[] usernames = {"user1", "user2", "user3", "user4"};
            string[] emails = {"user1@example.com", "user2@example.com", "user3@example.com", "user4@example.com"};
            string[] passwords = {"pass1", "pass2", "pass3", "pass4"};
            // create counters for accepted and rejected rows
            int rows = 0;
            int rejRows = 0;
            bool error = false;
            // Create the transaction
            VerticaTransaction txn = _conn.BeginTransaction();
            // Create the parameterized query and assign parameter types
            VerticaCommand command = _conn.CreateCommand();
            command.CommandText = "insert into TEST values (@id, @username, @email, @password)";
            command.Parameters.Add(new VerticaParameter("id", VerticaType.BigInt));
            command.Parameters.Add(new VerticaParameter("username", VerticaType.VarChar));
            command.Parameters.Add(new VerticaParameter("email", VerticaType.VarChar));
            command.Parameters.Add(new VerticaParameter("password", VerticaType.VarChar));
            // Prepare the statement
            command.Prepare();
            // Loop through the column arrays and insert the data
            for (int i = 0; i < ids.Length; i++)
            {
                command.Parameters["id"].Value = ids[i];
                command.Parameters["username"].Value = usernames[i];
                command.Parameters["email"].Value = emails[i];
                command.Parameters["password"].Value = passwords[i];
                try
                {
                    rows += command.ExecuteNonQuery();
                }
                catch (Exception e)
                {
                }
            }
            // Close the transaction
            txn.Rollback();
            // Close the connection
            _conn.Close();
        }
    }
}```
{
    Console.WriteLine("\nInsert failed - \n " + e.Message + "\n");
    ++rejRows;
    error = true;
}
}
if (error)
{
    // Roll back if errors
    Console.WriteLine("Errors. Rolling Back Transaction.");
    Console.WriteLine(rejRows + " rows rejected.");
    txn.Rollback();
}
else
{
    // Commit if no errors
    Console.WriteLine("No Errors. Committing Transaction.");
    txn.Commit();
    Console.WriteLine("Inserted " + rows + " rows. ");
}
_conn.Close();
}
Loading Batches Directly into ROS

When loading large batches of data (more than 100MB or so), you should load the data directly into ROS containers. Inserting directly into ROS is more efficient for large loads than AUTO mode, since it avoids overflowing the WOS and spilling the remainder of the batch to ROS. Otherwise, the Tuple Mover has to perform a moveout on the data in the WOS, while subsequent data is directly written into ROS containers. This results in the data from your batch being segmented across containers.

When you load data using AUTO mode, Vertica inserts the data first into the WOS. If the WOS is full, Vertica inserts the data directly into ROS. For details about load options, see Choosing a Load Method.

To directly load batches into ROS, set the DirectBatchInsert connection property to true. See Opening and Closing the Database Connection for details on all of the connection properties. When the DirectBatchInsert property is set to true, all batch inserts bypass the WOS and load directly into a ROS container.
Example usage:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            builder.DirectBatchInsert = true;
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            //Perform some operations
            _conn.Close();
        }
    }
}
```

Streaming Data Via ADO.NET

There are two options to stream data from a file on the client to your Vertica database through ADO.NET:

- Use the `VerticaCopyStream` ADO.NET class to stream data in an object-oriented manner
- Execute a `COPY LOCAL` SQL statement to stream the data

The topics in this section explain how to use these options.
Streaming From the Client Via VerticaCopyStream

The VerticaCopyStream class lets you stream data from the client system to a Vertica database. It lets you use the SQL COPY statement directly without having to copy the data to a host in the database cluster first by substituting one or more data stream(s) for STDIN.

Notes:

- Use Transactions and disable auto commit on the copy command for better performance.

- Disable auto commit using the copy command with the 'no commit' modifier. You must explicitly disable commits. Enabling transactions does not disable autocommit when using VerticaCopyStream.

- The copy command used with VerticaCopyStream uses copy syntax.

- VerticaCopyStream.rejects is zeroed every time execute is called. If you want to capture the number of rejects, assign the value of VerticaCopyStream.rejects to another variable before calling execute again.

- You can add multiple streams using multiple AddStream() calls.
Example usage:

The following example demonstrates using VerticaCopyStream to copy a file stream into Vertica.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.IO;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Configure connection properties
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";
            // Open the connection
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            try
            {
                using (_conn)
                {
                    // Start a transaction
                    VerticaTransaction txn = _conn.BeginTransaction();

                    // Create a table for this example
                    VerticaCommand command = new VerticaCommand("DROP TABLE IF EXISTS copy_table", _conn);
                    command.ExecuteNonQuery();
                    command.CommandText = "CREATE TABLE copy_table (Last_Name char(50), "
                                          + "First_Name char(50), Email char(50), "
                                          + "Phone_Number char(15))";
                    command.ExecuteNonQuery();
                    // Create a new filestream from the data file
                    string filename = "C:/customers.txt";
                    Console.WriteLine("\n\nLoading File: " + filename);
                    FileStream inputfile = File.OpenRead(filename);
                    // Define the copy command
                    string copy = "copy copy_table from stdin record terminator E\n" + "delimiter '|' + " enforcelength "
                        + " no commit";
                    // Create a new copy stream instance with the connection and copy statement
                    VerticaCopyStream vcs = new VerticaCopyStream(_conn, copy);

                    // Start the VerticaCopyStream process
                    vcs.Start();
                    // Add the file stream
                }
            }
        }
    }
}
```
vcs.AddStream(inputfile, false);

// Execute the copy
vcs.Execute();

// Finish stream and write out the list of inserted and rejected rows
long rowsInserted = vcs.Finish();
IList<long> rowsRejected = vcs.Rejects;
// Does not work when rejected or exceptions defined
Console.WriteLine("Number of Rows inserted: ", rowsInserted);  
Console.WriteLine("Number of Rows rejected: ", rowsRejected.Count);
if (rowsRejected.Count > 0)
    {
        for (int i = 0; i < rowsRejected.Count; i++)
            {
                Console.WriteLine("Rejected row #\{0\} is row \{1\}", i, rowsRejected[i]);
            }
    }

// Commit the changes
txn.Commit();
}
}

} catch (Exception e)
{
    Console.WriteLine(e.Message);
}

// close the connection
_conn.Close();
} }
Using Copy with ADO.NET

To use COPY with ADO.NET, just execute a COPY statement and the path to the source file on the client system. This method is simpler than using the VerticaCopyStream class. However, you may prefer using VerticaCopyStream if you have many files to copy to the database or if your data comes from a source other than a local file (streamed over a network connection, for example).

The following example code demonstrates using COPY to copy a file from the client to the database. It is the same as the code shown in Bulk Loading Using the COPY Statement and the path to the data file is on the client system, rather than on the server.

To load data that is stored on a database node, use a VerticaCommand object to create a COPY command:

1. **Create a connection to the database** through the node on which the data file is stored.
2. Create a command object using the connection.
   
   ```csharp
   VerticaCommand command = _conn.CreateCommand();
   ```
3. Copy data. The following is an example of using the COPY command to load data. It uses the LOCAL modifier to copy a file local to the client issuing the command.
   
   ```csharp
   command.CommandText = "copy lcopy_table from '/home/dbadmin/customers.txt'
   + " record terminator E'\n' delimiter '|'" + " enforcelength ";
   Int32 insertedRows = command.ExecuteNonQuery();
   Console.WriteLine(insertedRows + " inserted.");
   ```
Example Usage:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.IO;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Configure connection properties
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";

            // Open the connection
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();
            try
            {
                using (_conn)
                {
                    // Start a transaction
                    VerticaTransaction txn = _conn.BeginTransaction();

                    // Create a table for this example
                    VerticaCommand command = new VerticaCommand("DROP TABLE IF EXISTS lcopy_table");
                    command.ExecuteNonQuery();
                    command.CommandText = "CREATE TABLE IF NOT EXISTS lcopy_table (Last_Name char (50), "
                        + "First_Name char(50), Email char(50), "
                        + "Phone_Number char(15))";
                    command.ExecuteNonQuery();
                    // Define the copy command
                    command.CommandText = "copy lcopy_table from '/home/dbadmin/customers.txt'";
                    command.CommandText += " record terminator E'\n' delimiter '|'";
                    command.CommandText += " enforcelength "
                        + " no commit";
                    // Execute the copy
                    Int32 insertedRows = command.ExecuteNonQuery();
                    Console.WriteLine(insertedRows + " inserted.");
                    // Commit the changes
                    txn.Commit();
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.Message);
            }
        }
    }
}
```
Handling Messages (ADO.NET)

You can capture info and warning messages that Vertica provides to the ADO.NET driver by using the InfoMessage event on the VerticaConnection delegate class. This class captures messages that are not severe enough to force an exception to be triggered, but might still provide information that can benefit your application.

To Use the VerticalInfoMessageEventHandler class:

1. Create a method to handle the message sent from the even handler:

```csharp
static void conn_InfoMessage(object sender, VerticaInfoMessageEventArgs e)
{
    Console.WriteLine(e.SqlState + " : " + e.Message);
}
```

2. Create a connection and register a new VerticalInfoMessageHandler delegate for the InfoMessage event:

```csharp
_conn.InfoMessage += new VerticalInfoMessageEventHandler(conn_InfoMessage);
```

3. Execute your queries. If a message is generated, then the event handle function is run.

4. You can unsubscribe from the event with the following command:

```csharp
_conn.InfoMessage -= new VerticalInfoMessageEventHandler(conn_InfoMessage);
```

Example usage:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
```
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;
namespace ConsoleApplication
{
    class Program
    {

        // define message handler to deal with messages
        static void conn_InfoMessage(object sender, VerticaInfoMessageEventArgs e)
        {
            Console.WriteLine(e.SqlState + " : " + e.Message);
        }
        static void Main(string[] args)
        {

            //configure connection properties
            VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
            builder.Host = "192.168.1.10";
            builder.Database = "VMart";
            builder.User = "dbadmin";

            //open the connection
            VerticaConnection _conn = new VerticaConnection(builder.ToString());
            _conn.Open();

            //create message handler instance by subscribing it to the InfoMessage event of the connection
            _conn.InfoMessage += new VerticaInfoMessageEventHandler(conn_InfoMessage);

            //create and execute the command
            VerticaCommand cmd = _conn.CreateCommand();
            cmd.CommandText = "drop table if exists fakeTable";
            cmd.ExecuteNonQuery();

            //close the connection
            _conn.Close();
        }
    }
}

This examples displays the following when run:

00000: Nothing was dropped
Getting Table Metadata (ADO.Net)

You can get the table metadata by using the GetSchema() method on a connection and loading the metadata into a DataTable:

```csharp
DataTable table = _conn.GetSchema("Tables", new string[] { database_name, schema_name, table_name, table_type });
```

For example:
```csharp
DataTable table = _conn.GetSchema("Tables", new string[] { null, null, null, "SYSTEM TABLE" });
```

`database_name, schema_name, table_name` can be set to `null`, be a specific name, or use a LIKE pattern.

`table_type` can be one of:

- "SYSTEM TABLE"
- "TABLE"
- "GLOBAL TEMPORARY"
- "LOCAL TEMPORARY"
- "VIEW"
- null

If `table_type` is set to `null`, then the metadata for all metadata tables is returned.

Example Usage:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using Vertica.Data.VerticaClient;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            //configure connection properties
```
VerticaConnectionStringBuilder builder = new VerticaConnectionStringBuilder();
builder.Host = "192.168.1.10";
builder.Database = "VMart";
builder.User = "dbadmin";

//open the connection

VerticaConnection _conn = new VerticaConnection(builder.ToString());
_conn.Open();

//create a new data table containing the schema

//the last argument can be "SYSTEM TABLE", "TABLE", "GLOBAL TEMPORARY",
// "LOCAL TEMPORARY", "VIEW", or null for all types
DataTable table = _conn.GetSchema("Tables", new string[] { null, null, null, "SYSTEM TABLE" });

//print out the schema

foreach (DataRow row in table.Rows) {
    foreach (DataColumn col in table.Columns) {
        Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
    }
    Console.WriteLine("============================");
}

//close the connection

_conn.Close();
Programming Python Client Applications

To use Python with Vertica, you must either install the Vertica Python Client (see Python Client Documentation) or install the pyodbc module and a Vertica ODBC driver on the machine where Python is installed. See Python Prerequisites.

Python on Linux

Most Linux distributions come with Python preinstalled. If you want a more recent version, you can download and build it from the source code, though sometimes RPMs are also available. See the Python Web site and click an individual release for details. See also Python documentation.

To determine the Python version on your Linux operating systems, type the following at a command prompt:

```
# python -V
```

The system returns the version; for example:

```
Python 3.3.4
```

Python on Windows

Windows operating systems do not include Python by default. There are several different distributions of Python for windows:

- The ActiveState Web site distributes a free Windows installer for Python called ActivePython.

- The official Python.org site has installer packages for several versions of Python.

If you need installation instructions for Windows, see Using Python on Windows at python.org.
Python and Unicode

When you are using Python, be sure that all of your components are using the same unicode text encoding. By default, the DSN Parameter ColumnsAsChar causes the ODBC driver to report CHAR and VARCHAR values as SQL_WCHAR. The driver returns these values to the driver manager in the encoding expected by the driver manager, as controlled by the DriverManagerEncoding parameter in vertica.ini. Similarly, your Python application must use the encoding expected by the driver manager. If any of these components use different encodings, your output can become garbled.

The Vertica Python Client

Vertica has a native Python client you can use to communicate with your Vertica database. Before you can connect to Vertica using Python, you need to download the Python Client. See the Vertica Python Client Documentation, for download and installation instructions.

Using pyodbc and Vertica

Before you can connect to Vertica using pyodbc, you need to download the pyodbc module, which communicates with iODBC/unixODBC driver on UNIX operating systems and the ODBC Driver Manager for Windows operating systems.

The pyodbc module is an open source, MIT-licensed Python module, letting you use ODBC to connect to almost any database from Windows, Linux, Mac OS/X, and other operating systems. Vertica supports multiple versions of pyodbc. See Python Prerequisites for additional details.

Download the source distribution from the pyodbc Web site, unpack it and build it. Note that you need the unixODBC development package (in addition to the regular build tools) to build pyodbc. For example, on RedHat/CentOS run: yum install unixODBC-devel, and on Ubuntu run: sudo apt-get install unixodbc-dev. See the pyodbc wiki for detailed instructions.

Python Clients and the UUID Data Type

Both the Vertica Python client and Vertica ODBC driver (that pyodbc interacts with) do not support Vertica's native UUID data type. Values retrieved using these drivers from a
UUID column are converted to strings. When your client queries the metadata for a UUID column, the drivers report its data type as a string. Convert any UUID values that you want to insert into a UUID column to strings. Vertica automatically converts these values into the native UUID data type before inserting them into a table.

External Resources

- Python Database API Specification v2.0
- Python documentation

Configuring the ODBC Run-Time Environment on Linux

To configure the ODBC run-time environment on Linux:

1. Create the odbc.ini file if it does not already exist.
2. Add the ODBC driver directory to the LD_LIBRARY_PATH system environment variable:

   ```
   export LD_LIBRARY_PATH=/path-to-vertica-odbc-driver:$LD_LIBRARY_PATH
   ```

   **Important:** If you skip Step 2, the ODBC manager cannot find the driver in order to load it.

   These steps are relevant only for unixODBC and iODBC. See their respective documentation for details on odbc.ini.

See Also

- unixODBC Web site
- iODBC Web site

Querying the Database Using Python and pyodbc

The example session below uses pyodbc with the Vertica ODBC driver to connect Python to the Vertica database.
**Note:** SQLFetchScroll and SQLFetch functions cannot be mixed together in iODBC code. When using pyodbc with the iODBC driver manager, skip cannot be used with the fetchall, fetchone, and fetchmany functions.

### Example Script

The following example script shows how to query Vertica using Python 3, pyodbc, and an ODBC DSN.

```python
import pyodbc
cnxn = pyodbc.connect("DSN=VerticaDSN", ansi=True)
cursor = cnxn.cursor()
# create table
cursor.execute("CREATE TABLE TEST("
    "C_ID INT,"
    "C_FP FLOAT,"
    "C_VARCHAR VARCHAR(100),"
    "C_DATE DATE, C_TIME TIME,"
    "C_TS TIMESTAMP,"
    "C_BOOL BOOL")")
cursor.execute("INSERT INTO test VALUES(1,1.1,'abcdefg1234567890','1901-01-01','23:12:34','1901-01-01 09:00:09','t')")
cursor.execute("INSERT INTO test VALUES(2,3.4,'zxcasdqwe09876543','1991-11-11','00:00:01','1981-12-31 19:19:19','f')")
cursor.execute("SELECT * FROM TEST")
rows = cursor.fetchall()
for row in rows:
    print(row, end='\n')
cursor.execute("DROP TABLE TEST CASCADE")
cursor.close()
cnxn.close()
```

The resulting output displays:

```
(2, 3.4, 'zxcasdqwe09876543', datetime.date(1991, 11, 11), datetime.time(0, 0, 1), datetime.datetime(1981, 12, 31, 19, 19, 19), False)
(1, 1.1, 'abcdefg1234567890', datetime.date(1901, 1, 1), datetime.time(23, 12, 34), datetime.datetime(1901, 1, 1, 9, 0, 9), True)
```

### Notes

SQLPrimaryKeys returns the table name in the primary (pk_name) column for unnamed primary constraints. For example:
- **Unnamed primary key:**

```
CREATE TABLE schema.test(c INT PRIMARY KEY);
```

```text
SQLPrimaryKeys
"TABLE_CAT", "TABLE_SCHEM", "TABLE_NAME", "COLUMN_NAME", "KEY_SEQ", "PK_NAME" <Null>, "SCHEMA", "TEST", "C", 1, "TEST"
```

- **Named primary key:**

```
CREATE TABLE schema.test(c INT CONSTRAINT pk_1 PRIMARY KEY);
```

```text
SQLPrimaryKeys
"TABLE_CAT", "TABLE_SCHEM", "TABLE_NAME", "COLUMN_NAME", "KEY_SEQ", "PK_NAME" <Null>, "SCHEMA", "TEST", "C", 1, "PK_1"
```

[[[Undefined variable _Branding_Variables._Company_Acronym]]] recommends that you name your constraints.

## See Also

- [Loading Data Through ODBC](#)

## Programming Perl Client Applications

The Perl programming language has a Database Interface module (DBI) that creates a standard interface for Perl scripts to interact with databases. The interface module relies on Database Driver modules (DBDs) to handle all of the database-specific communication tasks. The result is an interface that provides a consistent way for Perl scripts to interact with many different types of databases.

Your Perl script can interact with Vertica using the Perl DBI module along with the DBD::ODBC database driver to interface to Vertica's ODBC driver. See the CPAN pages for Perl's [DBI](#) and [DBD::ODBC](#) modules for detailed documentation.
Important: With Perl ODBC clients, Vertica allows a forked process (a child process) to drop the parent connection to the Vertica server when the child process completes and exits. Vertica allows this behavior regardless of the setting of the Perl DBI AutoInactiveDestroy attribute. To change the default setting so that Vertica honors the setting of the Perl DBI AutoInactiveDestroy attribute, add the parameter CleanupInForkChild to your vertica.ini file, and set its value to 1. When the Perl DBI AutoInactiveDestroy attribute is set to 1, and the Vertica parameter CleanupInForkChild is set to 1, Vertica does not drop the parent connection upon child process completion.

Perl Client Prerequisites

In order run a Perl client script that connects to Vertica, your client system must have:

- The Vertica ODBC drivers installed and configured. See Installing the Vertica Client Drivers for details.
- A Data Source Name (DSN) containing the connection parameters for your Vertica. See Creating an ODBC Data Source Name. (Optionally, your Perl script can connect to Vertica without using a DSN as described in Connecting From Perl Without a DSN).
- A supported version of Perl installed
- The DBI and DBD::ODBC Perl modules (see below)

Supported Perl Versions

Vertica supports Perl versions 5.8 and 5.10. Versions later than 5.10 may also work.
Perl on Linux

Most Linux distributions come with Perl preinstalled. See your Linux distribution's documentation for details of installing and configuring its Perl package is it is not already installed.

To determine the Perl version on your Linux operating systems, type the following at a command prompt:

```
# perl -v
```

The system returns the version; for example:

```
This is perl, v5.10.0 built for x86_64-linux-thread-multi
```

Perl on Windows

Perl is not installed by default on Windows platforms. There are several different Perl packages you can download and install on your Windows system:

- **ActivePerl** by Activestate is a commercially-supported version of Perl for Windows platforms.
- **Strawberry Perl** is an open-source port of Perl for Windows.

The Perl Driver Modules (DBI and DBD::ODBC)

Before you can connect to Vertica using Perl, your Perl installation needs to have the Perl Database Interface module (DBI) and the Database Driver for ODBC (DBD::ODBC). These modules communicate with iODBC/unixODBC driver on UNIX operating systems or the ODBC Driver Manager for Windows operating systems.

Vertica supports the following Perl modules:

- **DBI version 1.609** (DBI-1.609.tar.gz)
- **DBD::ODBC version 1.22** (DBD-ODBC-1.22.tar.gz)

Later versions of DBI and DBD::ODBC may also work.

DBI is installed by default with many Perl installations. You can test whether it is installed by executing the following command on the Linux or Windows command line:
If the command exits without printing anything, then DBI is installed. If it prints an error, such as:

```
Can't locate DBI.pm in @INC (@INC contains:
  /usr/local/lib64/perl5/usr/local/share/perl5
  /usr/lib64/perl5/vendor_perl
  /usr/share/perl5/vendor_perl
  /usr/lib64/perl5
  /usr/share/perl5).
BEGIN failed--compilation aborted at -e line 1.
```

then DBI is not installed.

Similarly, you can see if DBD::ODBC is installed by executing the command:

```
# perl -e "use DBD::ODBC;"
```

You can also run the following Perl script to determine if DBI and DBD::ODBC are installed. If they are, the script lists any available DSNs.

```
#!/usr/bin/perl
use strict;
# Attempt to load the DBI module in an eval using require. Prevents
# script from erroring out if DBI is not installed.
 eval {
   require DBI;
   DBI->import();
};
if ($@) {
   # The eval failed, so DBI must not be installed
   print "DBI module is not installed\n";
} else {
   # Eval was successful, so DBI is installed
   print "DBI Module is installed\n";
   # List the drivers that DBI knows about.
   my @drivers = DBI->available_drivers;
   print "Available Drivers: \n";
   foreach my $driver (@drivers) {
      print "\t$driver\n";
   }
   # See if DBD::ODBC is installed by searching driver array.
   if (grep {/ODBC/} @drivers) {
      print "nDBD::ODBC is installed.\n";
      # List the ODBC data sources (DSNs) defined on the system
      print "Defined ODBC Data Sources:\n";
      my @dsns = DBI->data_sources('ODBC');
      foreach my $dsn (@dsns) {
         print "$dsn\n";
      }
   } else {
      print "DBD::ODBC is not installed\n";
   }
}
```
The exact output of the above code will depend on the configuration of your system. The following is an example of running the code on a Windows computer:

<table>
<thead>
<tr>
<th>DBI Module is installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Drivers:</td>
</tr>
<tr>
<td>- ADO</td>
</tr>
<tr>
<td>- DBM</td>
</tr>
<tr>
<td>- ExampleP</td>
</tr>
<tr>
<td>- File</td>
</tr>
<tr>
<td>- Gofer</td>
</tr>
<tr>
<td>- ODBC</td>
</tr>
<tr>
<td>- Pg</td>
</tr>
<tr>
<td>- Proxy</td>
</tr>
<tr>
<td>- SQLite</td>
</tr>
<tr>
<td>- Sponge</td>
</tr>
<tr>
<td>- mysql</td>
</tr>
</tbody>
</table>

| DBD::ODBC is installed.                  |
| Defined ODBC Data Sources:              |
| - dbi:ODBC:dBASE Files                  |
| - dbi:ODBC:Excel Files                  |
| - dbi:ODBC:MS Access Database           |
| - dbi:ODBC:VerticaDSN                   |

**Installing Missing Perl Modules**

If Perl's DBI or DBD::ODBC modules are not installed on your client system, you must install them before your Perl scripts can connect to Vertica. How you install modules depends on your Perl configuration:

- For most Perl installations, you use the `cpan` command to install modules. If the `cpan` command alias isn't installed on your system, you can try to start CPAN by using the command:

  ```bash
  perl -MCPAN -e shell
  ```

- Some Linux distributions provide Perl modules as packages that can be installed with the system package manager (such as `yum` or `apt`). See your Linux distribution's documentation for details.

- On ActiveState Perl for Windows, you use the Perl Package Manager (PPM) program to install Perl modules. See the [Activestate's PPM documentation](#) for details.

**Note:** Installing Perl modules usually requires administrator or root privileges. If you do not have these permissions on your client system, you need to ask your system administrator to install these modules for you.
Connecting to Vertica Using Perl

You use the Perl DBI module's connect function to connect to Vertica. This function takes a required data source string argument and optional arguments for the username, password, and connection attributes.

The data source string must start with "dbi:ODBC:”, which tells the DBI module to use the DBD::ODBC driver to connect to Vertica. The remainder of the string is interpreted by the DBD::ODBC driver. It usually contains the name of a DSN that contains the connection information needed to connect to your Vertica database. For example, to tell the DBD::ODBC driver to use the DSN named VerticaDSN, you use the data source string:

"dbi:ODBC:VerticaDSN"

The username and password parameters are optional. However, if you do not supply them (or just the username for a passwordless account) and they are not set in the DSN, attempting to connect always fails.

The connect function returns a database handle if it connects to Vertica. If it does not, it returns undef. In that case, you can access the DBI module's error string property ($DBI::errstr) to get the error message.

Note: By default, the DBI module prints an error message to STDERR whenever it encounters an error. If you prefer to display your own error messages or handle errors in some other manner, you may want to disable these automatic messages by setting DBI's PrintError connection attribute to false. See Setting Perl DBI Connection Attributes for details. Otherwise, users may see two error messages: the one that DBI prints automatically, and the one that your script prints on its own.

The following example demonstrates connecting to Vertica using a DSN named VerticaDSN. The call to connect supplies a username and password. After connecting, it calls the database handle's disconnect function, which closes the connection.

```perl
#!/usr/bin/perl -w
use strict;
use DBI;

# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr"
} else {
    print "Connected!\n";
    $dbh->disconnect();
```
Setting ODBC Connection Parameters in Perl

To set ODBC connection parameters, replace the DSN name with a semicolon delimited list of parameter name and value pairs in the source data string. Use the DSN parameter to tell DBD::ODBC which DSN to use, then add in other the other ODBC parameters you want to set. For example, the following code connects using a DSN named VerticaDSN and sets the connection's locale to en_GB.

```perl
#!/usr/bin/perl -w
use strict;
use DBI;

# Instead of just using the DSN name, use name and value pairs.
my $dbh = DBI->connect("dbi:ODBC:DSN=VerticaDSN;Locale=en_GB@collation=binary","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
print "Connected!\n";
$dbh->disconnect();
```

See Data Source Name (DSN) Connection Properties for a list of the connection parameters you can set in the source data string.

Setting Perl DBI Connection Attributes

The Perl DBI module has attributes that you can use to control the behavior of its database connection. These attributes are similar to the ODBC connection parameters (in several cases, they duplicate each other's functionality). The DBI connection attributes are a cross-platform way of controlling the behavior of the database connection.

You can set the DBI connection attributes when establishing a connection by passing the DBI `connect` function a hash containing attribute and value pairs. For example, to set the DBI connection attribute AutoCommit to false, you would use:

```perl
# Create a hash that holds attributes for the connection
my %attr = {AutoCommit => 0};
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
        %attr);
```

See the DBI documentation's Database Handle Attributes section for a full description of the attributes you can set on the database connection.

After your script has connected, it can access and modify the connection attributes through the database handle by using it as a hash reference. For example:
The following example demonstrates setting two connection attributes:

- **RaiseError** controls whether the DBI driver generates a Perl error if it encounters a database error. Usually, you set this to true (1) if you want your Perl script to exit if there is a database error.

- **AutoCommit** controls whether statements automatically commit their transactions when they complete. DBI defaults to Vertica's default AutoCommit value of true. Always set AutoCommit to false (0) when bulk loading data to increase database efficiency.

```perl
#!/usr/bin/perl
use strict;
use DBI;

# Create a hash that holds attributes for the connection
my $attr = {
    RaiseError => 1,  # Make database errors fatal to script
    AutoCommit => 0  # Prevent statements from committing
        # their transactions.
};

# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123", $attr);

if (defined $dbh->err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}

print "Connected!\n";

# The database handle lets you access the connection attributes directly:
print "The AutoCommit attribute is: " . $dbh->{AutoCommit} . "\n";
print "The RaiseError attribute is: " . $dbh->{RaiseError} . "\n";

# And you can change values, too...
$dbh->{AutoCommit} = 1;
print "The AutoCommit attribute is now: " . $dbh->{AutoCommit} . "\n";
$dbh->disconnect();
```

The example outputs the following when run:

```
Connected!The AutoCommit attribute is: 0
The RaiseError attribute is: 1
The AutoCommit attribute is now: 1
```

**Connecting From Perl Without a DSN**

If you do not want to set up a Data Source Name (DSN) for your database, you can supply all of the information Perl's DBD::ODBC driver requires to connect to your Vertica database in the data source string. This source string must the DRIVER= parameter that tells DBD::ODBC which
driver library to use in order to connect. The value for this parameter is the name assigned to the driver by the client system's driver manager:

- On Windows, the name assigned to the Vertica ODBC driver by the driver manager is Vertica.
- On Linux and other UNIX-like operating systems, the Vertica ODBC driver's name is assigned in the system's odbcinst.ini file. For example, if your /etc/odbcinst.ini contains the following:

```
[Vertica]
Description = Vertica ODBC Driver
Driver = /opt/vertica/lib64/libverticaodbc.so
```

you would use the name Vertica. See Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX for more information about the odbcinst.ini file.

You can take advantage of Perl's variable expansion within strings to use variables for most of the connection properties as the following example demonstrates.

```
#!/usr/bin/perl
use strict;
use DBI;
my $server='VerticaHost';
my $port = '5433';
my $database = 'VMart';
my $user = 'ExampleUser';
my $password = 'password123';
# Connect without a DSN by supplying all of the information for the connection.
# The DRIVER value on UNIX platforms depends on the entry in the odbcinst.ini file.
my $dbh = DBI->connect("dbi:ODBC:DRIVER={Vertica};Server=$server;" .
    "Port=$port;Database=$database;UID=$user;PWD=$password")
or die "Could not connect to database: " . DBI::errstr;
print "Connected!\n";
$dbh->disconnect();
```

**Note:** Surrounding the driver name with braces ({ and }) in the source string is optional.

### Executing Statements Using Perl

Once your Perl script has connected to Vertica (see Connecting to Vertica Using Perl), it can execute simple statements that return a value rather than a result set by using the Perl DBI module's do function. You usually use this function to execute DDL statements or data loading statements such as COPY (see Using COPY LOCAL to Load Data in Perl).
#!/usr/bin/perl
use strict;
use DBI;

# Disable autocommit
my $attr = {AutoCommit => 0};
# Open a connection using a DSN.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123", $attr);
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}

# You can use the do function to perform DDL commands.
# Drop any existing table.
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;");
# Create a table to hold data.
$dbh->do("CREATE TABLE TEST (
    C_ID INT, 
    C_FP FLOAT, 
    C_VARCHAR VARCHAR(100), 
    C_DATE DATE, C_TIME TIME, 
    C_TS TIMESTAMP, 
    C_BOOL BOOL)");

# Commit changes and exit.
$dbh->commit();
$dbh->disconnect();

Note: The do function returns the number of rows that were affected by the statement (or -1 if the count of rows doesn't apply or is unavailable). Usually, the only time you need to consult this value is after you deleted a number of rows or if you used a bulk load command such as COPY. You use other DBI functions instead of do to perform batch inserts and selects (see Batch Loading Data Using Perl and Querying Using Perl for details).

Batch Loading Data Using Perl

To load large batches of data into Vertica using Perl:

1. Set DBI's AutoCommit connection attribute to false to improve the batch load speed. See Setting Perl DBI Connection Attributes for an example of disabling AutoCommit.

2. Call the database handle's prepare function to prepare a SQL INSERT statement that contains placeholders for the data values you want to insert. For example:

```perl
# Prepare an INSERT statement for the test table
$sth = $dbh->prepare("INSERT into test values(?,?,?,?,?,?,?)");
```

The prepare function returns a statement handle that you will use to insert the data.
3. Assign data to the placeholders. There are several ways to do this. The easiest is to populate an array with a value for each placeholder in your INSERT statement.

4. Call the statement handle's execute function to insert a row of data into Vertica. The return value of this function call lets you know whether Vertica accepted or rejected the row.

5. Repeat steps 3 and 4 until you have loaded all of the data you need to load.

6. Call the database handle's commit function to commit the data you inserted.

The following example demonstrates inserting a small batch of data by populating an array of arrays with data, then looping through it and inserting each row.

```perl
#!/usr/bin/perl
use strict;
use DBI;

# Create a hash reference that holds a hash of parameters for the connection.
my $attr = {AutoCommit => 0, # Turn off autocommit
            PrintError => 0  # Turn off automatic error printing.
            # This is handled manually.
        };

# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
            $attr);

if (defined DBI::err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}

print "Connection AutoCommit state is: " . $dbh->{AutoCommit} . 

# Create table to hold inserted data
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;") or die "Could not drop table;"
$dbh->do("CREATE TABLE TEST ( 
    C_ID INT,
    C_FP FLOAT,
    C_VARCHAR VARCHAR(100),
    C_DATE DATE, C_TIME TIME,
    C_TS TIMESTAMP,
    C_BOOL BOOL)"") or die "Could not create table";

# Populate an array of arrays with values. One of these rows contains
# data that will not be successfully inserted. Another contains an
# undef value, which gets inserted into the database as a NULL.
my @data = (
    [1,1.111,'Hello World!','2001-01-01','01:01:01',
     '2001-01-01 01:01:01','t'],
    [2,2.22222,'How are you?','2002-02-02','02:02:02',
     '2002-02-02 02:02:02','f'],
    ['bad value',2.22222,'How are you?','2002-02-02','02:02:02',
     '2002-02-02 02:02:02','f'],
    [4,4.22222,undef,'2002-02-02','02:02:02',
     '2002-02-02 02:02:02','f'],
);

# Create a prepared statement to use parameters for inserting values.
my $sth = $dbh->prepare_catched("INSERT into test values(?,?,?,?,:,?)");
```
my $rowcount = 0; # Count # of rows
foreach my $tuple (@data) {
    $rowcount++;
    # Insert the row
    my $retval = $sth->execute(@$tuple);
    # See if the row was successfully inserted.
    if ($retval == 1) {
        # Value of 1 means the row was inserted (1 row was affected by insert)
        print "Row $rowcount successfully inserted\n";
    } else {
        print "Inserting row $rowcount failed";
        # Error message is not set on some platforms/versions of DBUI. Check to
        # ensure a message exists to avoid getting an uninitialized var warning.
        if ($sth->err()) {
            print ": ". $sth->errstr();
        } print "\n";
    }
}
# Commit changes. With AutoCommit off, you need to use commit for batched
# data to actually be committed into the database. If your Perl script exits
# without committing its data, Vertica rolls back the transaction and the
# data is not committed.
$dbh->commit();
$dbh->disconnect();

The previous example displays the following when successfully run:

Connection AutoCommit state is: 0
Row 1 successfully inserted
Row 2 successfully inserted
Inserting row 3 failed with error 01000 [Vertica][VerticaDSII] (20) An
error occurred during query execution: Row rejected by server; see
server log for details (SQL-01000)
Row 4 successfully inserted

Note that one of the rows was not inserted because it contained a string value that could not
be stored in an integer column. See Conversions Between Perl and Vertica Data Types for
details of data type handling in Perl scripts that communicate with Vertica.

Using COPY LOCAL to Load Data in Perl

If you have delimited files (for example, a file with comma-separated values) on your client
system that you want to load into Vertica, you can use the COPY LOCAL statement to directly
load the file’s contents into Vertica instead of using Perl to read, parse, and then batch insert
the data. You execute a COPY LOCAL statement to load the file from the local filesystem. The
result of executing the statement is the number of rows that were successfully inserted.

The following example code demonstrates loading a file named data.txt and located in the
same directory as the Perl file into Vertica using a COPY LOCAL statement.
#!/usr/bin/perl
use strict;
use DBI;
# Filesystem path handling module
use File::Spec;
# Create a hash reference that holds a hash of parameters for the
# connection.
my $attr = {AutoCommit => 0}; # Turn off AutoCommit
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123", $attr) or die "Failed to connect: $DBI::errstr";
print "Connected!
";
# Drop any existing table.
$dbh->do("DROP TABLE IF EXISTS Customers CASCADE;");
# Create a table to hold data.
$dbh->do("CREATE TABLE Customers( \\
    ID INT, \\
    FirstName VARCHAR(100),\\
    LastName VARCHAR(100),\\
    Email VARCHAR(100),\\
    Birthday DATE");
# Find the absolute path to the data file located in the current working
# directory and named data.txt
my $currDir = File::Spec->rel2abs(File::Spec->curdir());
my $dataFile = File::Spec->catfile($currDir, 'data.txt');
print "Loading file $dataFile
";
# Load local file using copy local. Return value is the # of rows affected
# which equates to the number of rows inserted.
my $rows = $dbh->do("COPY Customers FROM LOCAL 'dataFile' DIRECT")
or die $dbh->errstr;
print "Copied $rows rows into database.
";
$dbh->commit();
# Prepare a query to get the first 15 rows of the results
my $sth = $dbh->prepare("SELECT * FROM Customers WHERE ID < 15 \\
    ORDER BY ID");
$sth->execute() or die "Error querying table: ". $dbh->errstr;
my @row; # Pre-declare variable to hold result row used in format statement.
# Use Perl formats to pretty print the output. Declare the heading for the
# form.
format STDOUT_TOP =
ID First Last EMail Birthday
== ===== = ==== ===
# The Perl write statement will output a formatted line with values from the
# @row array. See http://perldoc.perl.org/perlform.html for details.
format STDOUT =
    @> @<<<<<<< @<<<<<<< @<<<<<<<<<<<<<<<<<<<<<<< @<<<<<<< @row
# Loop through result rows while we have them
while (@row = $sth->fetchrow_array()) {
    write; # Format command does the work of extracting the columns from
    # the @row array and writing them out to STDOUT.
}
# Call commit to prevent Perl from complaining about uncommitted transactions
# when disconnecting
$dbh->commit();
$dbh->disconnect();
The data.txt file is a text file containing a row of data on each line. The columns are delimited by pipe (|) characters. This is the default format that the COPY command accepts, which makes the COPY LOCAL statement in the example code simple. See the COPY statement entry in the SQL Reference Manual for handling data files that are in different formats. Here is an example of the content in this file:

<table>
<thead>
<tr>
<th>ID</th>
<th>First</th>
<th>Last</th>
<th>EMail</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Georgia</td>
<td>Gomez</td>
<td><a href="mailto:Rhiannon@magna.us">Rhiannon@magna.us</a></td>
<td>1937-10-03</td>
</tr>
<tr>
<td>2</td>
<td>Abdul</td>
<td>Alexander</td>
<td><a href="mailto:Kathleen@ipsum.gov">Kathleen@ipsum.gov</a></td>
<td>1941-03-10</td>
</tr>
<tr>
<td>3</td>
<td>Nigel</td>
<td>Contreras</td>
<td><a href="mailto:Tanner@et.com">Tanner@et.com</a></td>
<td>1955-06-01</td>
</tr>
<tr>
<td>4</td>
<td>Gray</td>
<td>Holt</td>
<td><a href="mailto:Thomas@Integer.us">Thomas@Integer.us</a></td>
<td>1945-12-06</td>
</tr>
<tr>
<td>5</td>
<td>Candace</td>
<td>Bullock</td>
<td><a href="mailto:Scott@vitae.gov">Scott@vitae.gov</a></td>
<td>1932-05-27</td>
</tr>
<tr>
<td>6</td>
<td>Matthew</td>
<td>Dotson</td>
<td><a href="mailto:Keith@Cras.com">Keith@Cras.com</a></td>
<td>1956-09-30</td>
</tr>
<tr>
<td>7</td>
<td>Haviva</td>
<td>Hopper</td>
<td><a href="mailto:Morgan@porttitor.edu">Morgan@porttitor.edu</a></td>
<td>1975-05-10</td>
</tr>
<tr>
<td>8</td>
<td>Stewart</td>
<td>Sweeney</td>
<td><a href="mailto:Rhonda@lectus.us">Rhonda@lectus.us</a></td>
<td>2003-06-20</td>
</tr>
<tr>
<td>9</td>
<td>Allen</td>
<td>Rogers</td>
<td><a href="mailto:Alexander@enim.gov">Alexander@enim.gov</a></td>
<td>2006-06-17</td>
</tr>
<tr>
<td>10</td>
<td>Trevor</td>
<td>Dillon</td>
<td><a href="mailto:Eagan@id.org">Eagan@id.org</a></td>
<td>1988-11-27</td>
</tr>
<tr>
<td>11</td>
<td>Leroy</td>
<td>Ashley</td>
<td><a href="mailto:Carter@turpis.edu">Carter@turpis.edu</a></td>
<td>1958-07-25</td>
</tr>
<tr>
<td>12</td>
<td>Elmo</td>
<td>Malone</td>
<td><a href="mailto:Carla@enim.edu">Carla@enim.edu</a></td>
<td>1978-08-29</td>
</tr>
<tr>
<td>13</td>
<td>Laurel</td>
<td>Ball</td>
<td><a href="mailto:Zelenia@Integer.us">Zelenia@Integer.us</a></td>
<td>1989-09-20</td>
</tr>
<tr>
<td>14</td>
<td>Zeus</td>
<td>Phillips</td>
<td><a href="mailto:Branden@blandit.gov">Branden@blandit.gov</a></td>
<td>1996-08-08</td>
</tr>
</tbody>
</table>

The example code produces the following output when run on a large sample file:

```
Connected!
Loading file /home/dbadmin/Perl/data.txt
Copied 1000000 rows into database.
```

<table>
<thead>
<tr>
<th>ID</th>
<th>First</th>
<th>Last</th>
<th>EMail</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Georgia</td>
<td>Gomez</td>
<td><a href="mailto:Rhiannon@magna.us">Rhiannon@magna.us</a></td>
<td>1937-10-03</td>
</tr>
<tr>
<td>2</td>
<td>Abdul</td>
<td>Alexander</td>
<td><a href="mailto:Kathleen@ipsum.gov">Kathleen@ipsum.gov</a></td>
<td>1941-03-10</td>
</tr>
<tr>
<td>3</td>
<td>Nigel</td>
<td>Contreras</td>
<td><a href="mailto:Tanner@et.com">Tanner@et.com</a></td>
<td>1955-06-01</td>
</tr>
<tr>
<td>4</td>
<td>Gray</td>
<td>Holt</td>
<td><a href="mailto:Thomas@Integer.us">Thomas@Integer.us</a></td>
<td>1945-12-06</td>
</tr>
<tr>
<td>5</td>
<td>Candace</td>
<td>Bullock</td>
<td><a href="mailto:Scott@vitae.gov">Scott@vitae.gov</a></td>
<td>1932-05-27</td>
</tr>
<tr>
<td>6</td>
<td>Matthew</td>
<td>Dotson</td>
<td><a href="mailto:Keith@Cras.com">Keith@Cras.com</a></td>
<td>1956-09-30</td>
</tr>
<tr>
<td>7</td>
<td>Haviva</td>
<td>Hopper</td>
<td><a href="mailto:Morgan@porttitor.edu">Morgan@porttitor.edu</a></td>
<td>1975-05-10</td>
</tr>
<tr>
<td>8</td>
<td>Stewart</td>
<td>Sweeney</td>
<td><a href="mailto:Rhonda@lectus.us">Rhonda@lectus.us</a></td>
<td>2003-06-20</td>
</tr>
<tr>
<td>9</td>
<td>Allen</td>
<td>Rogers</td>
<td><a href="mailto:Alexander@enim.gov">Alexander@enim.gov</a></td>
<td>2006-06-17</td>
</tr>
<tr>
<td>10</td>
<td>Trevor</td>
<td>Dillon</td>
<td><a href="mailto:Eagan@id.org">Eagan@id.org</a></td>
<td>1988-11-27</td>
</tr>
<tr>
<td>11</td>
<td>Leroy</td>
<td>Ashley</td>
<td><a href="mailto:Carter@turpis.edu">Carter@turpis.edu</a></td>
<td>1958-07-25</td>
</tr>
<tr>
<td>12</td>
<td>Elmo</td>
<td>Malone</td>
<td><a href="mailto:Carla@enim.edu">Carla@enim.edu</a></td>
<td>1978-08-29</td>
</tr>
<tr>
<td>13</td>
<td>Laurel</td>
<td>Ball</td>
<td><a href="mailto:Zelenia@Integer.us">Zelenia@Integer.us</a></td>
<td>1989-09-20</td>
</tr>
<tr>
<td>14</td>
<td>Zeus</td>
<td>Phillips</td>
<td><a href="mailto:Branden@blandit.gov">Branden@blandit.gov</a></td>
<td>1996-08-08</td>
</tr>
</tbody>
</table>

Note: Loading a single, large data file into Vertica through a single data connection is less efficient than loading a number of smaller files onto multiple nodes in parallel. Loading onto multiple nodes prevents any one node from becoming a bottleneck.

Querying Using Perl

To query Vertica using Perl:
1. Prepare a query statement using the Perl DBI module's `prepare` function. This function returns a statement handle that you use to execute the query and get the result set.

2. Execute the prepared statement by calling the `execute` function on the statement handle.

3. Retrieve the results of the query from the statement handle using one of several methods, such as calling the statement handle's `fetchrow_array` function to retrieve a row of data, or `fetchall_array` to get an array of arrays containing the entire result set (not a good idea if your result set may be very large!).

The following example demonstrates querying the table created by the example shown in **Batch Loading Data Using Perl**. It executes a query to retrieve all of the content of the table, then repeatedly calls the statement handle's `fetchrow_array` function to get rows of data in an array. It repeats this process until `fetchrow_array` returns `undef`, which means that there are no more rows to be read.

```perl
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1}; # Make errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123", $attr);
# Prepare a query to get the content of the table
my $sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
# Execute the query by calling execute on the statement handle
$sth->execute();
# Loop through result rows while we have them, getting each row as an array
while (my @row = $sth->fetchrow_array()) {
    # The @row array contains the column values for this row of data
    # Loop through the column values
    foreach my $column (@row) {
        if (!defined $column) {
            # NULLs are signaled by undefs. Set to NULL for clarity
            $column = "NULL";
        }
        print "$column\t"; # Output the column separated by a tab
    }
    print "\n";
}
$dbh->disconnect();
```

The example prints the following when run:

```
1 1.111 Hello World! 2001-01-01 01:01:01 2001-01-01 01:01:01 1
2 2.22222 How are you? 2002-02-02 02:02:02 2002-02-02 02:02:02 0
4 4.22222 NULL 2002-02-02 02:02:02 2002-02-02 02:02:02 0
```
Binding Variables to Column Values

Another method of retrieving the query results is to bind variables to columns in the result set using the statement handle's bind_columns function. You may find this method convenient if you need to perform extensive processing on the returned data, since your code can use variables rather than array references to access the data. The following example demonstrates binding variables to the result set, rather than looping through the row and column values.

```perl
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1};  # Make SQL errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN32","ExampleUser","password123", $attr);
# Prepare a query to get the content of the table
my $sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
$sth->execute();
# Create a set of variables to bind to the column values.
my ($C_ID, $C_FP, $C_VARCHAR, $C_DATE, $C_TIME, $C_TS, $C_BOOL);
# Bind the variable references to the columns in the result set.
$sth->bind_columns($C_ID, $C_FP, $C_VARCHAR, $C_DATE, $C_TIME, $C_TS, $C_BOOL);

# Now, calling fetch() to get a row of data updates the values of the bound
# variables. Continue calling fetch until it returns undefined.
while ($sth->fetch()) {
    # Note, you should always check that values are defined before using them,
    # since NULL values are translated into Perl as undefined. For this
    # example, just check the VARCHAR column for undefined values.
    if (!defined $C_VARCHAR) {
        $C_VARCHAR = "NULL";
    }
    # Just print values separated by tabs.
    print "$C_ID	$C_FP	$C_VARCHAR	$C_DATE	$C_TIME	$C_TS	$C_BOOL
";
}
$dbh->disconnect();
```

The output of this example is identical to the output of the previous example.

Preparing, Querying, and Returning a Single Row

If you expect a single row as the result of a query (for example, when you execute a `COUNT (*)` query), you can use the DBI module's selectrow_array function to combine executing a statement and retrieving an array as a result.

The following example shows using selectrow_array to execute and get the results of the `SHOW LOCALE` statement. It also demonstrates changing the locale using the do function.
#!/usr/bin/perl
use strict;
use DBI;
my $attr = {RaiseError => 1 }; # Make SQL errors fatal to the Perl script.
# Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);
# Demonstrate setting/getting locale.
# Use selectrow_array to combine preparing a statement, executing it, and
# getting an array as a result.
my @localerv = $dbh->selectrow_array("SHOW LOCALE;"autoplay);
# The locale name is the 2nd column (array index 1) in the result set.
print "Locale: $localerv[1]\n"autoplay;
# Use do() to execute a SQL statement to set the locale.
$dbh->do("SET LOCALE TO en_GB;"
autoplay);
# Get the locale again.
@localerv = $dbh->selectrow_array("SHOW LOCALE;"
autoplay);
print "Locale is now: $localerv[1]\n"autoplay;
$dbh->disconnect();

The result of running the example is:

Locale: en_US@collation=binary (LEN_KBINARY)
Locale is now: en_GB (LEN)

Executing Queries and ResultBufferSize Settings

When you call the execute() function on a prepared statement, the client library retrieves results up to the size of the result buffer. The result buffer size is set using ODBC's ResultBufferSize setting.

Vertica does not allow multiple active queries per connection. However, you can simulate multiple active queries by setting the result buffer to be large enough to accommodate the entire results from the first query. To ensure that the ODBC client driver’s buffer is large enough to store result set for first query you can set ResultBufferSize to 0. Setting this parameter to 0 makes the result buffer size unlimited. The ODBC driver allocates enough memory to read the entire result set. With the entire result set from the first query stored in the result set buffer, the database connection is free to perform another query. Your client can execute this second query even though it has not processed the entire result set from the first query.

However, if you set the ResultBufferSize to 0, you may find that your calls to execute() result in the operating system killing your Perl client script. The operating system may terminate your script if the ODBC driver allocates too much memory to store a large result set.

A workaround for this behavior is limit the number of rows returned by your query. Then you can set the ResultBufferSize to a value that accommodates this limited result set. For example, you can estimate the amount of memory needed to store a single row of your query result.
Then use the **LIMIT** and **OFFSET** clauses to get a specific number of rows that will fit into the space you allocated using ResultBufferSize. If the results of your query is able to fit within the limited result set buffer, you can then perform additional queries with the same database connection. This solution makes your code more complex as you will need to perform multiple queries to get the entire result set. Also, it is not appropriate in cases where you need to operate on an entire result set at once, rather than just a portion of it at a time.

A better solution is to use separate database connections for each query you want to perform. The overhead of the additional database connection is small compared to the resources needed to process large data sets.

**Conversions Between Perl and Vertica Data Types**

Perl is a loosely-typed programming language that does not assign specific data types to values. It converts between string and numeric values based on the operations being performed on the values. For this reason, Perl has little problem extracting most string and numeric data types from Vertica. All interval data types (DATE, TIMESTAMP, etc.) are converted to strings. You can use several different date and time handling Perl modules to manipulate these values in your scripts.

Vertica NULL values translate to Perl's undefined (undefined) value. When reading data from columns that can contain NULL values, you should always test whether a value is defined before using it.

When inserting data into Vertica, Perl's DBI module attempts to coerce the data into the correct format. By default, it assumes column values are VARCHAR unless it can determine that they are some other data type. If given a string value to insert into a column that has an integer or numeric data type, DBI attempts to convert the string's contents to the correct data type. If the entire string can be converted to a value of the appropriate data type, it inserts the value into the column. If not, inserting the row of data fails.

DBI transparently converts integer values into numeric or float values when inserting into column of FLOAT, NUMERIC, or similar data types. It converts numeric or floating values to integers only when there would be no loss of precision (the value to the right of the decimal point is 0). For example, it can insert the value 3.0 into an INTEGER column since there is no loss of precision when converting the value to an integer. It cannot insert 3.1 into an INTEGER column, since that would result in a loss of precision. It returns an error instead of truncating the value to 3.

The following example demonstrates some of the conversions that the DBI module performs when inserting data into Vertica.
#!/usr/bin/perl
use strict;
use DBI;
#
Create a hash reference that holds a hash of parameters for the
# connection.
my $attr = {AutoCommit => 0, # Turn off autocommit
    PrintError => 0, # Turn off print error. Manually handled
};
#
Open a connection using a DSN. Supply the username and password.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123",
    $attr);
if (defined DBI::err) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
print "Connection AutoCommit state is: " . $dbh->{AutoCommit} . "\n";
#
Create table to hold inserted data
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;");
$dbh->do("CREATE TABLE TEST (\n    C_ID INT, \
    C_FP FLOAT,\n    CVARCHAR VARCHAR(100),\n    CDATE DATE, C_TIME TIME,\n    C_TS TIMESTAMP,\n    C_BOOL BOOL)");
#
Populate an array of arrays with values.
my @data = {
    # Start with matching data types
    [1,1.111,'Matching datatypes','2001-01-01','01:01:01',
        '2001-01-01 01:01:01','t'],
    # Force floats -> int and int -> float.
    [2.0,2,"Ints <-> floats","2002-02-02","02:02:02",
        '2002-02-02 02:02:02',1],
    # Float -> int "only" works when there is no loss of precision.
    # this row will fail to insert:
    [3.1,3,"Float -> int with trunc?","2003-03-03","03:03:03",
        '2003-03-03 03:03:03',1],
    # String values are converted into numbers
    ["4","4.4","Strings -> numbers","2004-04-04","04:04:04",
        '2004-04-04 04:04:04',0],
    # String -> numbers only works if the entire string can be
    # converted into a number
    ["5 and a half","5.5","Strings -> numbers","2005-05-05",
        '05:05:05', '2005-05-05 05:05:05',0],
    # Number are converted into string values automatically,
    # assuming they fit into the column width.
    [6,6.6,3.14159, '2006-06-06','06:06:06',
        '2006-06-06 06:06:06',0],
    # There are some variations in the accepted date strings
    [7,7.7,'Date/time formats','07/07/2007','07:07:07',
        '07-07-2007 07:07:07',1],
};
#
Create a prepared statement to use parameters for inserting values.
my $sth = $dbh->prepare_cach ed("INSERT into test values(?,?,?,?,?,?,?)");
my $rowcount = 0; # Count # of rows
# Loop through the arrays to insert values
foreach my $tuple (@data) {
    $rowcount++;
    # Insert the row
    my $retval = $sth->execute($tuple);
}
# See if the row was successfully inserted.
if ($retval == 1) {
    # Value of 1 means the row was inserted (1 row was affected by insert)
    print "Row $rowcount successfully inserted\n";
} else {
    print "Inserting row $rowcount failed with error " .
    $sth->state . " " . $sth->errstr . "\n";
}
}
# Commit the data
$dbh->commit();
# Prepare a query to get the content of the table
$sth = $dbh->prepare("SELECT * FROM TEST ORDER BY C_ID ASC");
$sth->execute() or die "Error: " . $dbh->errstr;
my @row; # Need to pre-declare to use in the format statement.
# Use Perl formats to pretty print the output.
format STDOUT_TOP =
Int  Float VarChar  Date  Time  Timestamp  Bool
=== ====== ========= ====== ======= ======== ====
format STDOUT =
@>> @<<<< @<<<<<<<<<<<< @<<<<<< @<<<<<< @<<<<<<<<<<<< @<<<<
@row
# Loop through result rows while we have them
while (@row = $sth->fetchrow_array()) {
    write; # Format command does the work of extracting the columns from
    # the array.
}
# Commit to stop Perl complaining about in-progress transactions.
$dbh->commit();
$dbh->disconnect();

The example produces the following output when run:

Connection AutoCommit state is: 0
Row 1 successfully inserted
Row 2 successfully inserted
Inserting row 3 failed with error 01000 [Vertica][VerticaDSII] (20) An error occurred during query execution: Row rejected by server; see server log for details (SQL-01000)
Row 4 successfully inserted
Inserting row 5 failed with error 01000 [Vertica][VerticaDSII] (20) An error occurred during query execution: Row rejected by server; see server log for details (SQL-01000)
Row 6 successfully inserted
Row 7 successfully inserted
Int  Float VarChar  Date  Time  Timestamp  Bool
=== ====== ========= ====== ======= ======== ====
  1  1.111  Matching datatypes 2001-01-01 01:01:01 2001-01-01 01:01 1
  2   2  Ints <-> floats 2002-02-02 02:02:02 2002-02-02 02:02 1
  4  4.4  Strings -> numbers 2004-04-04 04:04:04 2004-04-04 04:04 0
  6  6.6  3.14159 2006-06-06 06:06:06 2006-06-06 06:06 0
  7  7.7  Date/time formats 2007-07-07 07:07:07 2007-07-07 07:07 1
Perl Unicode Support

Perl supports Unicode data with some caveats. See the perlunicode and the perlunitut (Perl Unicode tutorial) manual pages for details. (Be sure to see the copies of these manual pages included with the version of Perl installed on your client system, as the support for Unicode has changed in recent versions of Perl.) Perl DBI and DBD::ODBC also support Unicode, however DBD::ODBC must be compiled with Unicode support. See the DBD::ODBC documentation for details. You can check the DBD::ODBC-specific connection attribute named odbc_has_unicode to see if Unicode support is enabled in the driver.

The following example Perl script demonstrates directly inserting UTF-8 strings into Vertica and then reading them back. The example writes a text file with the output, since there are many problems displaying Unicode characters in terminal windows or consoles.

```perl
#!/usr/bin/perl
use strict;
use DBI;
# Open a connection using a DSN.
my $dbh = DBI->connect("dbi:ODBC:VerticaDSN","ExampleUser","password123");
unless (defined $dbh) {
    # Connection failed.
    die "Failed to connect: $DBI::errstr";
}
# Output to a file. Displaying Unicode characters to a console or terminal
# window has many problems. This outputs a UTF-8 text file that can
# be handled by many Unicode-aware text editors:
open OUTFILE, '>:utf8', "unicodeout.txt";
# See if the DBD::ODBC driver was compiled with Unicode support. If this returns
# 1, your Perl script will get get strings from the driver with the UTF-8
# flag set on them, ensuring that Perl handles them correctly.
print OUTFILE "Was DBD::ODBC compiled with Unicode support? ", $dbh->[odbc_has_unicode] . "\n";

# Create a table to hold VARCHARs
$dbh->do("DROP TABLE IF EXISTS TEST CASCADE;") ;

# Create a table to hold data. Remember that the width of the VARCHAR column
# is the number of bytes set aside to store strings, which often does not equal
# the number of characters it can hold when it comes to Unicode!
$dbh->do("CREATE TABLE test( C VARCHAR VARCHAR(100) )");
print OUTFILE "Inserting data...\n";
# Use Do to perform simple inserts
$dbh->do("INSERT INTO test VALUES('Hello')");
# This string contains several non-latin accented characters and symbols, encoded
# with Unicode escape notation. They are converted by Perl into UTF-8 characters
$dbh->do("INSERT INTO test VALUES('My favorite band is " . "\n\n{U+00E1}{U+00A4}{U+00FC}t \n{U+0062}{U+0065}{U+0072}{U+006F}{U+006C}{U+0061}{U+0072}{U+0069}{U+006E}{U+0065}k\n{U+0069}{U+006E}{U+0064}{U+0072}' )" );
# Some Chinese (Simplified) characters. This again uses escape sequence
# that Perl translates into UTF-8 characters.
$dbh->do("INSERT INTO test VALUES('x{4F60}\x{597D}')") ;
print OUTFILE "Getting data...\n";
```
# Prepare a query to get the content of the table
my $sth = $dbh->prepare_catched("SELECT * FROM test");
# Execute the query by calling execute on the statement handle
$sth->execute();
# Loop through result rows while we have them
while (my @row = $sth->fetchrow_array()) {
    # Loop through the column values
    foreach my $column (@row) {
        print OUTFILE "$column\t";
    }
    print OUTFILE "\n";
}
close OUTFILE;
$dbh->disconnect();

Viewing the unicodout.txt file in a UTF-8-capable text editor or viewer displays:

Was DBD::ODBC compiled with Unicode support? 1
Inserting data...
Getting data...
My favorite band is Ümläüt Överkïll Ø
你好
Hello

**Note:** Terminal windows and consoles often have problems properly displaying Unicode characters. That is why the example writes the output to a text file. With some text editors, you may need to manually set the encoding of the text file to UTF-8 in order for the characters to properly appear (and the font used to display text must have a full Unicode character set). If the character still do not show up, it may be that your version of DBD::ODBC was not compiled with UTF-8 support.

**See Also**

- [Unicode Character Encoding](#)
- [Required ODBC Driver Configuration Settings for Linux and UNIX](#)
Programming PHP Client Applications

You can connect to Vertica through PHP-ODBC using the Unix ODBC or iODBC library.

In order to use PHP with Vertica, you must install the following packages (and their dependencies):

- php
- php-odbc
- php-pdo
- UnixODBC (if you are using the Unix ODBC driver)
- libiodbc (if you are using the iODBC driver)

PHP on Linux

PHP is available with most Linux operating systems as a module for the Apache web server. Check your particular Linux repository for PHP RPMs or Debian packages. You can also build PHP from source. See the PHP web site for documentation and source downloads.

PHP on Windows

PHP is available for windows for both the Apache and IIS web servers. You can download PHP for Windows and view installation instructions at the PHP web site.

The PHP ODBC Drivers

PHP supports both the UnixODBC drivers and iODBC drivers. Both drivers use PHP's ODBC database abstraction layer.
Setup

You must read Programming ODBC Client Applications before connecting to Vertica through PHP. The following example ODBC configuration entries detail the typical settings required for PHP ODBC connections. The driver location assumes you have copied the Vertica drivers to /usr/lib64.

Example odbc.ini

```
[ODBC Data Sources]
VerticaDSNunixodbc = exampledb
VerticaDNSiodbc = exampledb2
[VerticaDSNunixodbc]
Description = VerticaDSN Unix ODBC driver
Driver = /usr/lib64/libverticaodbc.so
Database = Telecom
Servername = localhost
UserName = dbadmin
Password =
Port = 5433
[VerticaDNSiodbc]
Description = VerticaDSN iODBC driver
Driver = /usr/lib64/libverticaodbc.so
Database = Telecom
Servername = localhost
UserName = dbadmin
Password =
Port = 5433
```

Example odbcinst.ini

```
# Vertica
[VerticaDSNunixodbc]
Description = VerticaDSN Unix ODBC driver
Driver = /usr/lib64/libverticaodbc.so
[VerticaDNSiodbc]
Description = VerticaDSN iODBC driver
Driver = /usr/lib64/libverticaodbc.so
[ODBC]
Threading = 1
```
Verify the Vertica UnixODBC or iODBC Library

Verify the Vertica UnixODBC library can load all dependant libraries with the following command (assuming you have copies the libraries to /usr/lib64):

For example:

```
ldd /usr/lib64/libverticaodbc.so
```

You must resolve any "not found" libraries before continuing.

Test Your ODBC Connection

Test your ODBC connection with the following.

```
isql -v VerticaDSN
```

PHP Unicode Support

PHP does not offer native Unicode support. PHP only supports a 256-character set. However, PHP provides the UTF-8 functions `utf8_encode()` and `utf8_decode()` to provide some basic Unicode functionality.

See the PHP manual for strings for more details about PHP and Unicode.

Querying the Database Using PHP

The example script below details the use of PHP ODBC functions to connect to the Vertica Analytics Platform.

```php
<?php
    # Turn on error reporting
    error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
    # A simple function to trap errors from queries
    function errortrap_odbc($conn, $sql) {
        if (!$rs = odbc_exec($conn,$sql)) {
            echo "<br/>Failed to execute SQL: $sql<br/>". odbc_errormsg($conn);
        } else {
            echo "<br/>Success: " . $sql;
        }
        return $rs;
?>
```
Example Output

The following is the example output from the script.
Managing Query Execution Between the Client and Vertica

Multiple Active Result Sets (MARS)

ResultBufferSize

ResultBufferSize

By default, Vertica uses the ResultBufferSize parameter to determine the maximum size (in bytes) of a result set that a client can retrieve from a server. When ResultBufferSize is enabled, Vertica sends rows of data directly to the client making the query. The number of rows returned to the client at each fetch of data depends on the size (in bytes) of the ResultBufferSize parameter.

Sometimes, the size of the result set requested by the client is greater than what the ResultBufferSize parameter allows. In such cases, Vertica retrieves only a portion of the result set at a time. Each fetch of data returns the amount of data equal to the size set by the ResultBufferSize parameter. Ultimately, as the client iterates over the individual fetches of data, the entire result set is returned.
Benefits of ResultBufferSize

If you are concerned with the effect of your queries on network latency, ResultBufferSize may provide an advantage over MARS. MARS requires that the client wait until all rows of data are written to the server before the client can retrieve the data. This delay may cause latency issues for your network while waiting for the results to be stored.

In addition, MARS requires that you send two separate requests to return rows of data. The first request performs the query execution which stores the result set on the server. The second request retrieves the data rows that are stored on the server. With ResultBufferSize, you only need to send one request. This request both executes and retrieves the data rows of interest.

Query Execution with ResultBufferSize

The following graphic shows how Vertica returns rows of data from a database to the client with ResultBufferSize enabled:
The query execution performs the following steps:

1. The client sends a query, such as a `SELECT` statement, to the server. In the preceding graphic, the first query is named Query 1.
2. The server receives the client's request and begins to send both a description of the result set and the requested rows of data back to the client.

3. After all possible rows are returned to the client, the execution is complete. The size of the data set returned equals either that of the data that was requested or the maximum amount of data that ResultBufferSize parameter can retrieve. If the ResultBufferSize maximum size is not yet reached, Vertica can execute Query 2.

The server can accept Query 2 and perform the same steps that it did for Query 1. If the results for Query 1 had reached the maximum ResultBufferSize allowable, Vertica could not execute Query 2 until the client freed the results from Query 1.

After Query 2 runs, you cannot view the results you retrieved for Query 1, unless you execute Query 1 again.

Setting an Unlimited Buffer Size

Setting ResultBufferSize to 0 tells the client driver to use an unlimited result set buffer. With this setting, the client library allocates as much memory as it needs to read the entire result set of a query. You may choose to set ResultBufferSize to 0 if you want to simulate having multiple active queries over a single database connection at the same time. With an unlimited buffer size, your client can run a query and have its entire result set stored in memory. This ends the first query, so your client can execute a second query before it fully processes the results of the first query.

A drawback of this method is that your query may consume too much memory if your queries return large result sets. This over-allocation of memory can result in the operating system terminating your client. Due to this risk, consider using multiple database connections instead of trying to reuse a single connection for multiple queries. The overhead of multiple database connections is small compared to the overall amount of resources required to process a large data set.

Multiple Active Result Sets (MARS)

You can only enable MARS when you connect to Vertica using a JDBC client connection. MARS allows the execution of multiple queries on a single connection. While ResultBufferSize sends the results of a query directly to the client, MARS stores the results first on the server. Once query execution has finished and all of the results have been stored, you can make a retrieval request to the server to have rows returned to the client.
MARS is set at the session level and must be enabled for every new session. When MARS is enabled, ResultBufferSize is disabled. No error is returned, however the ResultBufferSize parameter is ignored.

Benefits of MARS

In comparison with ResultBufferSize, MARS enables you to store multiple result sets from different queries at the same time. You can also send new queries before all of the results of a previous result set have been returned to the client. This allows applications to decouple query execution from result retrieval so that, on a single connection, you can process different results at the same time.

When you enable ResultBufferSize, you must wait until all result sets have been returned to the client before a new query can be executed.

Another benefit of MARS is that it allows you to free up query resources faster than ResultBufferSize allows. While a query is running, resources are held by that query session. When ResultBufferSize is enabled, a client that is performing slowly might read a single row of a result set and then have to stop to retrieve the next row. This prevents the query from finishing quickly and, therefore, prevents the resources used from being freed up for other applications. With MARS, the speed of the client is irrelevant to the reading of rows. As soon as the results are written to the MARS storage, the resources are freed and the speed at which the client retrieves rows no longer matters.

Query Execution with MARS

The following graphic demonstrates how multiple queries to the server are handled when MARS is enabled:
Query 1:

1. Query 1 is sent to the server.

2. Query 1's row description and the status of its result set are returned to the client. However, no results are returned to the client at this time.

3. Query 1 completes and its results are saved on the server.
a. You can now send commands to retrieve the rows of Query 1's result set. These rows are stored on the server. Retrieved rows are sent to the client along with the status of the result set. By keeping track of the status of the result set, Vertica is able to keep track of which rows have been retrieved from the server.

4. Now that Query 1 has successfully completed, and its result sets are being stored on the server, Query 2 can be executed.

Query 2:

1. Query 2 is sent to the server.
2. Query 2's row description and the status of its result set are returned to the client. However, no results are returned to the client at this time.
3. Query 2 completes and its results are stored on the server. Both Query 1 and Query 2 now have result sets stored on the server.
4. You can now send retrieval requests to both Query 1 and Query 2’s result sets that are stored on the server. Whenever a retrieval request is made for rows from Query 1, the request is sent and rows and the result set status are sent to the client. The same occurs for Query 2.

Once all rows have been read by the client, the MARS storage on the server closes the active results session. The MARS storage on the server is then freed to store more data. The MARS storage also closes and frees once your session is finished.

Enabling and Disabling MARS

You can enable and disable MARS in two different ways:

1. To enable MARS using the JDBC client connection properties, see JDBC Connection Properties.

2. To enable MARS using the SET SESSION command, see SET SESSION MULTIPLEACTIVERESULTSETS.

See Also

- SESSION_MARS_STORE
- CLOSE_RESULTSET
• CLOSE_ALL_RESULTSETS
Management API

The Management API is a REST API that you can use to view and manage Vertica databases with scripts or applications that accept REST and JSON. The response format for all requests is JSON.

cURL

cURL is a command-line tool and application library used to transfer data to or from a server. It uses URL syntax, such as HTTP and HTTPS. All API requests sent to a Vertica server must be made using HTTPS. A request made using HTTP will fail.

There are four HTTP requests that can be passed using cURL to call API methods:

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieves data.</td>
</tr>
<tr>
<td>PUT</td>
<td>Updates data.</td>
</tr>
<tr>
<td>POST</td>
<td>Creates new data.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes data.</td>
</tr>
</tbody>
</table>

Syntax

curl https://<NODE>:5444/

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h --help</td>
<td>Lists all available command-line options.</td>
</tr>
<tr>
<td>-H --header</td>
<td>Allows you to use custom headers in your command. This is useful for sending a request that requires a VerticaAPIKEY.</td>
</tr>
</tbody>
</table>

curl -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/
**-k --insecure**

Allows SSL connections without certificate validation.

```
curl -k https://<NODE>:5444/
```

**-X --request**

Specifies the custom request used when communicating with a server.

```
curl -X REQUEST https://<NODE>:5444/
```

Can be one of the following values:

- GET
- PUT
- POST
- DELETE

**Note:** If no request is specified, cURL automatically defaults to the GET request.

There are many more options available to add to your cURL query. For a comprehensive list with descriptions, visit the [cURL Documentation Website](#).

## General API Information

These API calls can interact with either standard Vertica nodes or Management Console nodes.

<table>
<thead>
<tr>
<th>GET /</th>
<th>Returns the agent-specific information useful for version checking and service discovery.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET api</td>
<td>Returns a list of api objects and properties.</td>
</tr>
</tbody>
</table>
GET /

Returns API version information and a list of links to child resources for the Management API.

Resource URL

https://<NODE>:5444/

Authentication

Not required.

Parameters

None.

Example Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>https://&lt;NODE&gt;:5444/</td>
</tr>
</tbody>
</table>

Response:

```json
{
  "body": {
    "mime-types": [
      "default",
      "application/vertica.database.configuration.json-v2",
      "application/json",
      "application/vertica.nodes.json-v2",
      "default",
      "application/json",
      "default",
      "application/vertica.jobs.json-v2",
      "default",
      "application/vertica.hosts.json-v2",
      "application/json",
      "default",
      "application/vertica.hosts.json-v2",
      "application/json",
      "default"
    ]
  }
}
```
"application/json",
"application/vertica.host.json-v2",
"default",
"application/vertica.hosts.json-v2",
"application/json",
"application/vertica.nodes.json-v2",
"default",
"application/json",
"default",
"application/json",
"application/vertica.database.json-v2",
"default",
"application/vertica.hosts.json-v2",
"application/json",
"default",
"application/vertica.hosts.json-v2",
"application/json",
"default",
"application/json",
"application/vertica.databases.json-v2",
"application/vertica.nodes.json-v2",
"default",
"application/json",
"application/vertica.agent.json-v2",
"default",
"application/json",
"default",
"application/vertica.users.json-v2",
"application/json"
],

"version": "7.1.0"
},

"href": "/",

"links": [
"/databases",
"/hosts",
"/nodes",
"/licenses",
"/webhooks",
"/backups",
"/restore",
"/jobs"
],

"mime-type": "application/vertica.agent.json-v2"
GET api

Displays a list of all Management API commands and a brief description for each command. You can also access an HTML version of this information at https://<NODE>:5444/api.html. Replace <NODE> with the URL of a cluster node.

Resource URL

https://<NODE>:5444/api

Authentication

Not required.

Parameters

None.

Example Request

<table>
<thead>
<tr>
<th>GET</th>
<th>https://&lt;NODE&gt;:5444/api</th>
</tr>
</thead>
</table>

Response:

[
  {
    "accepts": {},
    "description": "Returns the agent specific information useful for version checking and service discovery",
    "method": "GET",
    "params": [],
    "route": "/
  },
  {
    "accepts": {},
    "description": "build the list of cluster objects and properties and return it as a JSON formatted array",
    "method": "GET",
    "params": [],
    "route": "/api"
  }]
"accepts": {},
"description": " list all the backups that have been created for all vbr configuration files ( *.ini ) that are located in the /opt/vertica/config directory.",
"method": "GET",
"params": [],
"route": "/backups"
},

{ "accepts": {},
"description": " create a new backup as defined by the given vbr configuration script base (filename minus the .ini extension) ",
"method": "POST",
"params": [],
"route": "/backups/:config_script_base"
},

{ "accepts": {},
"description": " get the detail for a specific backup archive ",
"method": "GET",
"params": [],
"route": "/backups/:config_script_base/:archive_id"
},

{ "accepts": {},
"description": " delete a backup based on the config ini file script",
"method": "DELETE",
"params": [],
"route": "/backups/:config_script_base/:backup_id"
},

{ "accepts": {},
"description": " build the list of databases, their properties, and current status ( from cache ) and return it as a JSON formatted array ",
"method": "GET",
"params": [],
"route": "/databases"
},

{ "accepts": {},
"description": " Create a new database by supplying a valid set of parameters ",
"method": "POST",
"params": [
  "name": name of the database to create",
  "passwd": password used by the database administrative user",
  "only": optional list of hostnames to include in database",
  "exclude": optional list of hostnames to exclude from the database",
  "catalog": directory used for the vertica catalog",
  "data": directory used for the initial vertica storage location",
  "port": port the database will listen on (default 5433)"
],
"route": "/databases"
},

{ "accepts": {},
"description": " Retrieve the database properties structure ",
"method": "GET",
"params": [],
"route": "/databases/:database_name"
Control / alter a database values using the PUT http method 

- **Method:** PUT
- **Params:**
  - action : value one of start|stop|rebalance|wla
- **Route:** "databases/:database_name"
"method": "DELETE",
"params": [],
"route": "/databases/:database_name/hosts/:host_id"
},

{ "accepts": {},
"description": " start the database process on a specific host participating in this
database. ",
"method": "POST",
"params": [],
"route": "/databases/:database_name/hosts/:host_id/process"
},

{ "accepts": {},
"description": " run the stop action against the given database for a specific host",
"method": "DELETE",
"params": [],
"route": "/databases/:database_name/hosts/:host_id/process"
},

{ "accepts": {},
"description": " replace a host with a standby host in the database ",
"method": "POST",
"params": [
  "user_id : vertica database username",
  "passwd : vertica database password"
],
"route": "/databases/:database_name/hosts/:host_id/replace_with/:host_id_new"
},

{ "accepts": {},
"description": " return the vertica license that this database is using ",
"method": "GET",
"params": [
  "user_id : vertica database user",
  "passwd : vertica database password"
],
"route": "/databases/:database_name/license"
},

{ "accepts": {},
"description": " this method upgrades the license in the database by using the license in
/opt/vertica/config/share ",
"method": "PUT",
"params": [
  "user_id : vertica database user",
  "passwd : vertica database password"
],
"route": "/databases/:database_name/license"
},

{ "accepts": {},
"description": " return all the feature licenses that this database is using ",
"method": "GET",
"params": [
  "user_id : vertica database user",
  "passwd : vertica database password"
],
"route": "/databases/:database_name/licenses"
},


{ "accepts": {},
  "description": "build the list of nodes for a given database, their properties, and current status (from cache) and return it as a JSON formatted array",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/nodes"
},
{ "accepts": {},
  "description": "build the list of nodes for a given database, their properties, and current status and return it as a JSON formatted array",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/nodes/:node_id"
},
{ "accepts": {},
  "description": "run the start action against the given database",
  "method": "POST",
  "params": [
    "epoch": start the database from this epoch",
    "include": start the database on these hosts only"
  ],
  "route": "/databases/:database_name/process"
},
{ "accepts": {},
  "description": "easy way to see if a database is running -- returns state of UP or DOWN",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/process"
},
{ "accepts": {},
  "description": "run the stop action against the given database",
  "method": "DELETE",
  "params": [
    "user_id": vertica database username",
    "passwd": vertica database password"
  ],
  "route": "/databases/:database_name/process"
},
{ "accepts": {},
  "description": "run the rebalance data action against the given database -- could be very long running!
  "method": "POST",
  "params": [
    "user_id": vertica database username",
    "passwd": vertica database password"
  ],
  "route": "/databases/:database_name/rebalance/process"
},
{ "accepts": {},
  "description": "Retrieve the database properties structure",
  "method": "GET",
  "params": [],
  "route": "/databases/:database_name/status"}


```
{
  "accepts": {},
  "description": "run the analyze workload action against the given database - could be very long running!",
  "method": "POST",
  "params": [
    "user_id": "vertica database username",
    "passwd": "vertica database password"
  ],
  "route": "/databases/:database_name/wla/process"
},
{
  "accepts": {},
  "description": "build a list of nodes in the cluster independent of their database associations",
  "method": "GET",
  "params": [],
  "route": "/hosts"
},
{
  "accepts": {},
  "description": "lists the properties of a given host in the cluster by calling the RESTful service on that agent!",
  "method": "GET",
  "params": [],
  "route": "/hosts/:hostid"
},
{
  "accepts": {},
  "description": "Returns a list of jobs the agent is tracking along with their current status and exit codes!",
  "method": "GET",
  "params": [],
  "route": "/jobs"
},
{
  "accepts": {},
  "description": "Deletes a specific job by canceling any outstanding activity associated with it!",
  "method": "DELETE",
  "params": [],
  "route": "/jobs/:id"
},
{
  "accepts": {},
  "description": "Returns the details (the saved output) for a specific job!",
  "method": "GET",
  "params": [],
  "route": "/jobs/:id"
},
{
  "accepts": {},
  "description": "POST your vertica license to this url using HTTP file upload format!",
  "method": "POST",
  "params": [
    "license": "the file to upload (use html form post format)"
  ],
  "route": "/licenses"
}
```
Rest APIs for the Agent

These API calls interact with standard Vertica nodes.
## Backup and Restore

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET backups</strong></td>
<td>Returns all the backups that have been created for all vbr configuration files (<code>.ini</code>) that are located in the <code>/opt/vertica/config</code> directory.</td>
</tr>
<tr>
<td><strong>POST backups/:config_script_base</strong></td>
<td>Creates a new backup as defined by the given vbr configuration script base (filename without the <code>.ini</code> extension).</td>
</tr>
<tr>
<td><strong>GET backups/:config_script_base/:archive_id</strong></td>
<td>Returns details for a specific backup archive.</td>
</tr>
<tr>
<td><strong>POST restore/:archive_id</strong></td>
<td>Restores a backup.</td>
</tr>
</tbody>
</table>

## Databases

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET databases</strong></td>
<td>Returns a list of databases, their properties, and current status.</td>
</tr>
<tr>
<td><strong>POST databases</strong></td>
<td>Creates a new database by supplying a valid set of parameters.</td>
</tr>
<tr>
<td><strong>GET databases/:database_name</strong></td>
<td>Returns details about a specific database.</td>
</tr>
<tr>
<td><strong>PUT databases/:database_name</strong></td>
<td>Starts, stops, rebalances, or runs Workload Analyzer on a database.</td>
</tr>
<tr>
<td><strong>DELETE databases/:database_name</strong></td>
<td>Deletes an existing database.</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/configuration</strong></td>
<td>Returns the current configuration parameters from the database.</td>
</tr>
<tr>
<td><strong>PUT databases/:database_name/configuration</strong></td>
<td>Sets one or more configuration parameters in the database.</td>
</tr>
<tr>
<td><strong>GET databases/:database_namehosts</strong></td>
<td>Returns hosts details for a specific database.</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/hosts</strong></td>
<td>Adds a new host to the database.</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>DELETE databases/:database_name/hosts/:host_id</strong></td>
<td>Removes a host from the database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/hosts/:host_id/process</strong></td>
<td>Starts the database process on a specific host.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>DELETE databases/:database_name/hosts/:host_id/process</strong></td>
<td>Stops the database on a specific host.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/hosts/:host_id/replace_with/:host_id_new</strong></td>
<td>Replaces a host with a standby host in the database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/license</strong></td>
<td>Returns the Vertica license that the specified database is using.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>PUT databases/:database_name/license</strong></td>
<td>Upgrades the license in the database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/licenses</strong></td>
<td>Returns all the feature licenses that the specified database is using.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/nodes</strong></td>
<td>Returns a list of nodes for the specified database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/nodes/:node_id</strong></td>
<td>Returns details on a specific node for the specified database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/process</strong></td>
<td>Starts the specified database.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>GET databases/:database_name/process</strong></td>
<td>Returns the state of the database as either UP or DOWN.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>DELETE databases/:database_name/process</strong></td>
<td>Stops the specified database on all hosts.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/rebalance/process</strong></td>
<td>Rebalances the specified database. This option can have a long run time.</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>POST databases/:database_name/wla/process</strong></td>
<td>Runs the analyze workload action against the specified database. This option can have a long run time.</td>
</tr>
</tbody>
</table>
**Hosts**

<table>
<thead>
<tr>
<th>GET hosts</th>
<th>Returns a list of hosts in this cluster.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET hosts/:hostid</td>
<td>Returns details for a specific host in this cluster.</td>
</tr>
</tbody>
</table>

**Jobs**

<table>
<thead>
<tr>
<th>GET jobs</th>
<th>Returns a list of jobs the agent is tracking, along with their current status and exit codes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET jobs/:id</td>
<td>Returns the details (the saved output) for a specific job.</td>
</tr>
</tbody>
</table>

**Licenses**

<table>
<thead>
<tr>
<th>POST licenses</th>
<th>Uploads and applies a new license to this cluster.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET licenses</td>
<td>Returns the license field that databases created on this cluster use.</td>
</tr>
</tbody>
</table>

**Nodes**

<table>
<thead>
<tr>
<th>GET nodes</th>
<th>Returns a list of nodes in this cluster.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET nodes/:nodeid</td>
<td>Returns details for a specific node in this cluster.</td>
</tr>
</tbody>
</table>
## Webhooks

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>webhooks</td>
<td>Returns a list of active webhooks.</td>
</tr>
<tr>
<td>POST</td>
<td>webhooks/subscribe</td>
<td>Creates a new webhook.</td>
</tr>
<tr>
<td>DELETE</td>
<td>webhooks/:subscriber_id</td>
<td>Deletes an existing webhook.</td>
</tr>
</tbody>
</table>
VerticaAPIClave

The Management API requires an authentication key, named VerticaAPIClave, to access some API resources. You can manage API keys by using the apikeymgr command-line tool.

usage: apikeymgr [-h] [--user REQUESTOR] [--app APPLICATION] [--delete]  
[--create] [--update] [--migrate]  
[--secure {restricted,normal,admin}] [--list]

API key management tool

optional arguments:  
-h, --help show this help message and exit
--user REQUESTOR The name of the person requesting the key
--app APPLICATION The name of the application that will use the key
--delete Delete the key for the given R & A
--create Create a key for the given R & A
--update Update a key for the given R & A
--migrate migrate the keyset to the latest format
--secure {restricted,normal,admin} Set the keys security level
--list List all the keys known

Example Request

To create a new VerticaAPIClave for the dbadmin user with admin access, enter the following:

```
$ apikeymgr --user dbadmin --app vertica --create --secure admin
```

Response:

Requestor : dbadmin
Application: vertica
API Key : ValidAPIKey
Synchronizing cluster...

Backup and Restore

You can use these API calls to perform backup and restore tasks for your database.

| GET backups | Returns all the backups that have been created for all vbr configuration files ( *.ini ) that are located in the /opt/vertica/config directory. |

Vertica Analytic Database (9.0.x)  Page 4807 of 6180
<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>backups/:config_script_base</td>
<td>Creates a new backup as defined by the given vbr configuration script base (filename without the .ini extension).</td>
</tr>
<tr>
<td>GET</td>
<td>backups/:config_script_base/:archive_id</td>
<td>Returns details for a specific backup archive.</td>
</tr>
<tr>
<td>POST</td>
<td>restore/:archive_id</td>
<td>Restores a backup.</td>
</tr>
</tbody>
</table>
GET backups

Returns a list of all backups created for vbr configuration (*.ini) files that reside in /opt/vertica/config and provides details about each backup.

Resource URL

https://<NODE>:5444/backups

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/backups |

Response:

```
[
  {
    "backups": [
      {
        "backup": "backup3_20140707_114852",
        "epoch": "61",
        "hosts": "v_vmart_node0001(10.20.100.247)",
        "objects": ""
      }
    ],
    "config_file": "/opt/vertica/config/backup3.ini",
    "num_backups": 1
  },
  {
    "backups": [
      {
        "backup": "backup_20140707_113737",
        "epoch": "61",
        "hosts": "v_vmart_node0001(10.20.100.247)",
        "objects": ""
      }
    ],
    "config_file": "/opt/vertica/config/backup2.ini",
    "num_backups": 1
  }
]
```
"objects": "",
},
{
"backup": "backup_archive20140707_113645",
"epoch": "60",
"hosts": "v_vmart_node0001(10.20.100.247)",
"objects": ""
}
],
"config_file": "/opt/vertica/config/backup.ini",
"num_backups": 2
}
POST backups/:config_script_base

Creates a new backup job for the backup defined in the vbr configuration script :config_script_base. The vbr configuration script must reside in /opt/vertica/configuration. The :config_script_base value does not include the .ini filename extension.

To determine valid :config_script_base values, see GET backups.

Returns a job ID that you can use to determine the status of the job.

Resource URL

https://<NODE>:5444/backups/:config_script_base

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

| POST | https://<NODE>:5444/backups/backup3 |

Response:

```json
{
  "id": "CreateBackup-VMart-1404750602.03",
  "url": "/jobs/CreateBackup-VMart-1404750602.03"
}
```
GET backups/:config_script_base/:archive_id

Returns details on a specific backup. You must provide the :config_script_base. This value is the name of a vbr config file (without the .ini filename extension) that resides in /opt/vertica/config. The :archive_id is the value of the backup field that the GET backups command returns.

Resource URL

https://<NODE>:5444/backups/:config_script_base/:archive_id

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET   | https://<NODE>:5444/backups/backup3/backup3_20140707_123254 |

Response:

```json
{
   "backup": "backup3_20140707_123254",
   "config_file": "/opt/vertica/config/backup3.ini",
   "epoch": "62",
   "hosts": "v_vmart_node0001(10.20.100.247)",
   "objects": ""
}
```
POST restore/:archive_id

Creates a new restore job to restore the database from the backup archive identified by :archive_id. The :archive_id is the value of a backup field that the GET backups command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/restore/:archive_id

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

| POST | https://<NODE>:5444/restore/backup3_20140707_132904 |

Response:

```json
{
  "id": "RestoreBackup-VMart-1404760113.71",
  "url": "/jobs/RestoreBackup-VMart-1404760113.71"
}
```

Databases

You can use these API calls to interact with your database.

<p>| GETdatabases | Returns a list of databases, their properties, and current status. |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>databases</td>
<td>Creates a new database by supplying a valid set of parameters.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name</td>
<td>Returns details about a specific database.</td>
</tr>
<tr>
<td>PUT</td>
<td>databases/:database_name</td>
<td>Starts, stops, rebalances, or runs Workload Analyzer on a database.</td>
</tr>
<tr>
<td>DELETE</td>
<td>databases/:database_name</td>
<td>Deletes an existing database.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name/configuration</td>
<td>Returns the current configuration parameters from the database.</td>
</tr>
<tr>
<td>PUT</td>
<td>databases/:database_name/configuration</td>
<td>Sets one or more configuration parameters in the database.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name.hosts</td>
<td>Returns hosts details for a specific database.</td>
</tr>
<tr>
<td>POST</td>
<td>databases/:database_name.hosts</td>
<td>Adds a new host to the database.</td>
</tr>
<tr>
<td>DELETE</td>
<td>databases/:database_name.hosts/:host_id</td>
<td>Removes a host from the database.</td>
</tr>
<tr>
<td>POST</td>
<td>databases/:database_name.hosts/:host_id/process</td>
<td>Starts the database process on a specific host.</td>
</tr>
<tr>
<td>DELETE</td>
<td>databases/:database_name.hosts/:host_id/process</td>
<td>Stops the database on a specific host.</td>
</tr>
<tr>
<td>POST</td>
<td>databases/:database_name.hosts/:host_id/replace_with/:host_id_new</td>
<td>Replaces a host with a standby host in the database.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name/license</td>
<td>Returns the Vertica license that the specified database is using.</td>
</tr>
<tr>
<td>PUT</td>
<td>databases/:database_name/license</td>
<td>Upgrades the license in the database.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name/licenses</td>
<td>Returns all the feature licenses that the specified database is using.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name/nodes</td>
<td>Returns a list of nodes for the specified database.</td>
</tr>
<tr>
<td>GET</td>
<td>databases/:database_name/nodes/:node_id</td>
<td>Returns details on a specific node for the specified database.</td>
</tr>
<tr>
<td>Endpoint</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>POST databases/:database_name/process</td>
<td>Starts the specified database.</td>
<td></td>
</tr>
<tr>
<td>GET databases/:database_name/process</td>
<td>Returns the state of the database as either UP or DOWN.</td>
<td></td>
</tr>
<tr>
<td>DELETE databases/:database_name/process</td>
<td>Stops the specified database on all hosts.</td>
<td></td>
</tr>
<tr>
<td>POST databases/:database_name/rebalance/process</td>
<td>Rebalances the specified database. This option can have a long run time.</td>
<td></td>
</tr>
<tr>
<td>POST databases/:database_name/wla/process</td>
<td>Runs the analyze workload action against the specified database. This option can have a long run time.</td>
<td></td>
</tr>
</tbody>
</table>
GETdatabases

Returns a list of databases, their current status, and database properties.

Resource URL

https://<NODE>:5444/databases

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

<table>
<thead>
<tr>
<th>GET</th>
<th>https://&lt;NODE&gt;:5444/databases</th>
</tr>
</thead>
</table>

An example of the full request using cURL:

curl -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/databases

Response:

```json
{
  "body": [
    {
      "href": "/databases/VMart",
      "mime-type": [
        "application/vertica.database.json-v2"
      ],
      "name": "VMart",
      "port": "5433",
      "status": "UP"
    },
    {
      "href": "/databases/testDB",
      "mime-type": [
        "application/vertica.database.json-v2"
      ],
    }
  ]
}```
"name": "testDB",
"port": "5433",
"status": "DOWN"
}
],
"href": "/databases",
"links": [
"/:database_name"
],
"mime-type": "application/vertica.databases.json-v2"}
POST databases

Creates a job to create a new database with the provided parameters.

Important: You must stop any running databases on the nodes on which you want to create the new database. If you do not, database creation fails.

Returns a job ID that can be used to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/database

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have admin level security.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the database to create.</td>
</tr>
<tr>
<td>passwd</td>
<td>Password for the new database.</td>
</tr>
<tr>
<td>only</td>
<td>Optional list of hostnames to include in the database. By default, all nodes in the cluster are added to the database.</td>
</tr>
<tr>
<td>exclude</td>
<td>Optional list of hostnames to exclude from the database.</td>
</tr>
<tr>
<td>catalog</td>
<td>Path of the catalog directory.</td>
</tr>
<tr>
<td>data</td>
<td>Path of the data directory.</td>
</tr>
<tr>
<td>port</td>
<td>Port where the database listens for client connections. Default is 5433.</td>
</tr>
</tbody>
</table>

Example Request

```
POST https://<NODE>:5444/databases?passwd=vertica&name=testDB
&catalog=%2Fhome%2Fdbadmin%2FtestDB
&data=%2Fhome%2Fdbadmin%2FtestDB
```
Response:

{
   "userid": "dbadmin"
}
GET databases/:database_name

Returns details about a specific database. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/databases/:database_name

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

<table>
<thead>
<tr>
<th>GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>https://&lt;NODE&gt;:5444/databases/VMart</td>
</tr>
</tbody>
</table>

Response:

```json
{
    "body": {
        "database_id": "VMart",
        "id": "VMart",
        "nodes": "v_vmart_node0001,v_vmart_node0002,v_vmart_node0003",
        "nodes_new": [
            {
                "catalog_base": "/home/dbadmin",
                "data_base": "/home/dbadmin",
                "host": "10.20.100.247",
                "id": "v_vmart_node0001"
            },
            {
                "catalog_base": "/home/dbadmin",
                "data_base": "/home/dbadmin",
                "host": "10.20.100.248",
                "id": "v_vmart_node0002"
            },
            {
```
"catalog_base": "/home/dbadmin",
"data_base": "/home/dbadmin",
"host": "10.20.100.249",
"id": "v_vmart_node0003"
"path": "/home/dbadmin/VMart",
"port": "5433",
"restartpolicy": "ksafe",
"status": "UP"
"href": "/databases/VMart",
"links": [
  "/configuration",
  "/hosts",
  "/license",
  "/nodes",
  "/process",
  "/rebalance/process",
  "/status",
  "/wla/process"
"mime-type": "application/vertica.database.json-v2"}
PUT databases/:database_name

Creates a job to run the action specified by the action parameter against the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have normal level security or higher.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>A password for the username.</td>
</tr>
<tr>
<td>action</td>
<td>Can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• start — Start the database.</td>
</tr>
<tr>
<td></td>
<td>• stop — Stop the database.</td>
</tr>
<tr>
<td></td>
<td>• rebalance — Rebalance the database.</td>
</tr>
<tr>
<td></td>
<td>• wla — Run Work Load Analyzer against the database.</td>
</tr>
</tbody>
</table>

Example Request

| PUT | https://<NODE>:5444/databases/testDB?user_id=dbadmin"&"passwd=vertica"&"action=stop |

Response:
DELETE databases/:database_name

Creates a job to delete (drop) an existing database on the cluster. To perform this operation, you must first stop the database. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have admin level security.

Parameters

None.

Example Request

```
DELETE

https://<NODE>:5444/databases/TestDB
```

Response:

```
{
  "id": "DropDatabase-TestDB-2014-07-18 12:50:33.332383",
  "url": "/jobs/DropDatabase-TestDB-2014-07-18 12:50:33.332383"
}
```
GET databases/:database_name/configuration

Returns a list of configuration parameters for the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/databases/:database_name/configuration

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>user_id</th>
<th>A database username.</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
</tbody>
</table>

Example Request

| GET         | https://<NODE>:5444/databases/testDB/configuration |

Response:

This API call returns over 100 configuration parameters. The following response is a small subset of the total amount returned.

```json
[
  {
    "catalog_value": "1",
    "change_requires_restart": "f",
    "change_under_support_guidance": "f",
    "current_value": "1",
    "database_value": "1",
    "default_value": "1",
    "description": "No of active partitions",
    "groups": "",
    "is_mismatch": "f",
    "node_name": "ALL",
```
"parameter_name": "ActivePartitionCount",
"source": "DEFAULT"
},
{
  "catalog_value": "180",
  "change_requires_restart": "f",
  "change_under_support_guidance": "f",
  "current_value": "180",
  "database_value": "180",
  "default_value": "180",
  "description": "Interval between advancing the AHM (seconds)",
  "groups": "",
  "is_mismatch": "f",
  "node_name": "ALL",
  "parameter_name": "AdvanceAHMInterval",
  "source": "DEFAULT"
},
{
  "catalog_value": "3",
  "change_requires_restart": "f",
  "change_under_support_guidance": "f",
  "current_value": "3",
  "database_value": "3",
  "default_value": "3",
  "description": "HDFS replication factor for Vertica data",
  "groups": "",
  "is_mismatch": "f",
  "node_name": "ALL",
  "parameter_name": "HadoopFSReplication",
  "source": "DEFAULT"
},
{
  "catalog_value": "",
  "change_requires_restart": "f",
  "change_under_support_guidance": "f",
  "current_value": "",
  "database_value": "",
  "default_value": "",
  "description": "Path to the java binary for executing UDx written in Java",
  "groups": "",
  "is_mismatch": "f",
  "node_name": "ALL",
  "parameter_name": "JavaBinaryForUDx",
  "source": "DEFAULT"
},
{
  "catalog_value": "80",
  "change_requires_restart": "f",
  "change_under_support_guidance": "f",
  "current_value": "80",
  "database_value": "80",
  "default_value": "80",
  "description": "The max amount of memory TopK(Heap) can use in MB",
  "groups": "",
  "is_mismatch": "f",
  "node_name": "ALL",
  "parameter_name": "TopKHeapMaxMem",
  "source": "DEFAULT"
}
}
"catalog_value": "READ COMMITTED",
"change_requires_restart": "f",
"change_under_support_guidance": "f",
"current_value": "READ COMMITTED",
"database_value": "READ COMMITTED",
"default_value": "READ COMMITTED",
"description": "READ COMMITTED (Default) - Last epoch for reads and current epoch for writes.
SERIALIZABLE - Current epoch for reads and writes",
"groups": "",
"is_mismatch": "f",
"node_name": "ALL",
"parameter_name": "TransactionIsolationLevel",
"source": "DEFAULT"
PUT databases/:database_name/configuration

Sets one or more configuration parameters for the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns the parameter name, the requested value, and the result of the attempted change (Success or Failed).

Resource URL

https://<NODE>:5444/databases/:database_name/configuration

Authentication

Requires a VerticaAPIKey in the request header. The API key must have admin level security.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
<tr>
<td>parameter_name</td>
<td>A parameter name and value combination for the parameter to be changed. Values must be URL encoded. You can include multiple name/value pairs to set multiple parameters with a single API call.</td>
</tr>
</tbody>
</table>

Example Request

```
PUT https://<NODE>:5444/databases/testDB/configuration?user_id=dbadmin"&"passwd=vertica
&"JavaBinaryForUDx=%2Fusr%2Fbin%2Fjava"&"TransactionIsolationLevel=SERIALIZABLE
```

Response:

```
[
  {
    "key": "JavaBinaryForUDx",
    "result": "Success",
    "value": "/usr/bin/java"
  }
]
```
{  "key": "TransactionIsolationLevel",  "result": "Success",  "value": "SERIALIZABLE"}
]
GET databases/:database_name/hosts

Returns the hostname/IP address, node name, and UP/DOWN status of each host associated with the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/databases/:database_name/hosts

Authentication

Requires a VerticaAPIKey in the request header. The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/databases/VMart/hosts |

Response:

```
{
  "body": [
    {
      "hostname": "10.20.100.247",
      "nodename": "v_vmart_node0001",
      "status": "UP",
    },
    {
      "hostname": "10.20.100.248",
      "nodename": "v_vmart_node0002",
      "status": "UP",
      "ts": "2014-07-18T13:12:31.904209"
    },
    {
      "hostname": "10.20.100.249",
      "nodename": "v_vmart_node0003",
      "status": "UP",
      "ts": "2014-07-18T13:12:31.904209"
    }
  ]
}
POST databases/:database_name/hosts

Creates a job to add a host to the database identified by :database_name. This host must already be part of the cluster. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

**Resource URL**

https://<NODE>:5444/:database_name/hosts

**Authentication**

Requires a VerticaAPIKey in the request header.

The API key must have admin level security.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
<tr>
<td>hostname</td>
<td>The hostname to add to the database. This host must already be part of the cluster.</td>
</tr>
</tbody>
</table>

**Example Request**

```
```

**Response:**

```
{
   "id": "AddHostToDatabase-testDB-2014-07-20 12:24:04.088812",
   "url": "/jobs/AddHostToDatabase-testDB-2014-07-20 12:24:04.088812"
}
```
DELETE databases/:database_name/hosts/:host_id

Creates a job to remove the host identified by :host_id from the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns. The :host_id is the value of the host field returned by GET databases/:database_name.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/hosts/:host_id

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have admin level security.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>A password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
</table>

Response:

```json
{
   "id": "RemoveHostFromDatabase-testDB-2014-07-20 13:41:15.646235",
   "url": "/jobs/RemoveHostFromDatabase-testDB-2014-07-20 13:41:15.646235"
}
```
POST databases/:database_name/hosts/:host_id/process

Creates a job to start the vertica process for the database identified by :database_name on the host identified by :host_id. The :database_name is the value of the name field that the GETdatabases command returns. The :host_id is the value of the host field returned by GET databases/:database_name.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/:database_name/hosts/:host_id/process

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request


Response:

```json
{
   "id": "StartDatabase-testDB-2014-07-20 13:14:03.968340",
   "url": "/jobs/StartDatabase-testDB-2014-07-20 13:14:03.968340"
}
```
GET databases/:database_name/license

Returns details about the database license being used by the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/:database_name/license

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>https://&lt;NODE&gt;:5444/VMart/license</td>
</tr>
</tbody>
</table>

Response:

```
{
    "body": {
        "details": {
            "assigned_to": "Vertica Systems, Inc.",
            "grace_period": 0,
            "is_ce": false,
            "is_unlimited": false,
            "name": "vertica",
            "not_after": "Perpetual",
            "not_before": "2007-08-03"
        },
        "last_audit": {
            "database_size_bytes": "814060522"
        }
    }
}
```
"license_size_bytes": "536870912000",
"usage_percent": "0.00151630588248372"
],
"href": "/databases/VMart/license",
"links": [],
"mime-type": "application/vertica.license.json-v2"}
GET databases/:database_name/licenses

Returns details about all license being used by the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/:database_name/licenses

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>user_id</th>
<th>A database username.</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
</tbody>
</table>

Example Request

GET

https://<NODE>:5444/VMart/licenses

Response:

```json
{
  "body": [
    {
      "details": {
        "assigned_to": "Vertica Systems, Inc.",
        "audit_date": "2014-07-19 21:35:25.111312",
        "is_ce": "False",
        "name": "vertica",
        "noderestriction": "",
        "not_after": "Perpetual",
        "not_before": "2007-08-03",
        "size": "500GB"
      },
      "last_audit": {
```
"database_size_bytes": "819066288",
"license_size_bytes": "536870912000",
"usage_percent": "0.00152562984824181"
}
},
{
  "details": {
    "assigned_to": "Vertica Systems, Inc., FlexTable",
    "audit_date": "2014-07-19 21:35:25.111312",
    "is_ce": "False",
    "name": "com.vertica.flextable",
    "node_restriction": "",
    "not_after": "Perpetual",
    "not_before": "2007-08-03",
    "size": "500GB"
  },
  "last_audit": {
    "audit_date": "2014-07-19 21:35:25.111312",
    "database_size_bytes": 0,
    "license_size_bytes": 536870912000,
    "usage_percent": 0
  }
},
"href": "/databases/VMart/licenses",
"links": [],
"mime-type": "application/vertica.features.json-v2"
DELETE databases/:database_name/hosts/:host_id/process

Creates a job to stop the vertica process for the database identified by :database_name on the host identified by :host_id. The :database_name is the value of the name field that the GETdatabases command returns. The :host_id is the value of the host field returned by GET databases/:database_name.

Returns a job ID that can be used to determine the status of the job. See GET jobs.

Note: If stopping the database on the hosts causes the database to no longer be k-safe, then the all database nodes may shut down.

Resource URL

https://<NODE>:5444/databases/:database_name/hosts/:host_id/process

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

DELETE


Response:

```json
{
  "id": "StopDatabase-testDB-2014-07-20 13:02:08.453547",
  "url": "/jobs/StopDatabase-testDB-2014-07-20 13:02:08.453547"
}
```
POST databases/:database_name/hosts/:host_id/replace_with/:host_id_new

Creates a job to replace the host identified by hosts/:host_id with the host identified by replace_with/:host_id. Vertica performs these operations for the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns. The :host_id is the value of the host field as returned by GET databases/:database_name. You can find valid replacement hosts using GET hosts. The replacement host cannot already be part of the database. You must stop the vertica process on the host being replaced.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/hosts/:host_id/replace_with/:host_id_new

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have admin level security.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>A password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>POS</th>
<th></th>
</tr>
</thead>
</table>

Response:

```json
{
}
```
GET databases/:database_name/nodes

Returns a comma-separated list of node IDs for the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/:database_name/nodes

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/VMart/nodes |

Response:

[
  {
    "database_id": "VMart",
    "node_id": "v_vmart_node0001,v_vmart_node0002,v_vmart_node0003",
    "status": "Unknown"
  }
]
GET databases/:database_name/nodes/:node_id

Returns details about the node identified by :node_id. The :node_id is one of the node IDs returned by GET databases/:database_name/nodes.

Resource URL

https://<NODE>:5444/:database_name/nodes/:node_id

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/databases/VMart/nodes/v_vmart_node0001 |

Response:

```
{
    "db": "VMart",
    "host": "10.20.100.247",
    "name": "v_vmart_node0001",
    "state": "UP"
}
```
POST databases/:database_name/process

Creates a job to start the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that can be used to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/process

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epoch</td>
<td>Start the database from this epoch.</td>
</tr>
<tr>
<td>include</td>
<td>Include only these hosts when starting the database. Use a comma-separated list of hostnames.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>https://&lt;NODE&gt;:5444/databases/:testDB/process</td>
</tr>
</tbody>
</table>

An example of the full request using cURL:

curl -X POST -H "VerticaApikey: ValidAPIKey" https://<NODE>:5444/:testDB/process

Response:

```json
{
  "id": "StartDatabase-testDB-2014-07-20 12:41:46.061408",
  "url": "/jobs/StartDatabase-testDB-2014-07-20 12:41:46.061408"
}
```
GET databases/:database_name/process

Returns a state of UP or DOWN for the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Resource URL

https://<NODE>:5444/databases/:database_name/process

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

Example Request

GET

<table>
<thead>
<tr>
<th>GET</th>
<th>https://&lt;NODE&gt;:5444/databases/VMart/process</th>
</tr>
</thead>
</table>

Response:

```
{
  "state": "UP"
}
```
DELETE databases/:database_name/process

Creates a job to stop the database identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/process

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>The password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>METHOD</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>https://&lt;NODE&gt;:5444/databases/testDB/process?user_id=dbadmin&quot;&amp;&quot;passwd=vertica</td>
</tr>
</tbody>
</table>

An example of the full request using cURL:

curl -X DELETE -H "VerticaApiKey: ValidAPIKey" https://<NODE>:5444/:testDB/process?user_id=dbadmin"&"passwd=vertica

Response:

```
{
  "id": "StopDatabase-testDB-2014-07-20 12:46:04.486637",
  "url": "/jobs/StopDatabase-testDB-2014-07-20 12:46:04.486637"
}
```
POST databases/:database_name/rebalance/process

Creates a job to run a rebalance on the database identified by host identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/rebalance/process

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>A database username.</td>
</tr>
<tr>
<td>passwd</td>
<td>A password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>method</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>https://&lt;NODE&gt;:5444/databases/testDB/rebalance/process</td>
</tr>
</tbody>
</table>

Response:

```json
{
    "id": "RebalanceData-testDB-2014-07-20 21:42:45.731038",
    "url": "/jobs/RebalanceData-testDB-2014-07-20 21:42:45.731038"
}
```
POST databases/:database_name/wla/process

Creates a job to run Workload Analyzer on the database identified by host identified by :database_name. The :database_name is the value of the name field that the GETdatabases command returns.

Returns a job ID that you can use to determine the status of the job. See GET jobs.

Resource URL

https://<NODE>:5444/databases/:database_name/wla/process

Authentication

Requires a VerticaAPKey in the request header.
The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>user_id</th>
<th>A database username.</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>A password for the username.</td>
</tr>
</tbody>
</table>

Example Request

<table>
<thead>
<tr>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>https://&lt;NODE&gt;:5444/databases/testDB/wla/process</td>
</tr>
</tbody>
</table>

Response:

```json
{
}
```

Hosts

You can use these API calls to get information on the hosts in your cluster.
<table>
<thead>
<tr>
<th>GET hosts</th>
<th>Returns a list of hosts in this cluster.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET hosts/:hostid</td>
<td>Returns details for a specific host in this cluster.</td>
</tr>
</tbody>
</table>
GET hosts

Returns a list of the hosts in the cluster and the hardware, software, and network details about each host.

Resource URL

https://<NODE>:5444/hosts

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/hosts |

Response:

```json
{
   "body": [
      {
         "cpu_info": {
            "cpu_type": "Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
            "number_of_cpus": 2
         },
         "host_id": "10.20.100.247",
         "hostname": "v_vmart_node0001.example.com",
         "max_user_proc": "3833",
         "nics": [
            {
               "broadcast": "10.20.100.255",
               "ipaddr": "10.20.100.247",
               "name": "eth0",
               "netmask": "255.255.255.0",
               "speed": "unknown"
            },
            {
            }
         ]
      }
   ]
}
```
"broadcast": "255.255.255.255",
"ipaddr": "127.0.0.1",
"name": "lo",
"netmask": "255.0.0.0",
"speed": "locallink"
},
"total_memory": 3833,
"vertica": {
"arch": "x86_64",
"brand": "vertica",
"release": "20140716",
"version": "9.0.0"
}
},
{ "cpu_info": {
"cpu_type": "Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
"number_of_cpus": 2
},
"host_id": "10.20.100.248",
"hostname": "v_vmart_node0002.example.com",
"max_user_proc": "3833",
"nics": [
{ "broadcast": "10.20.100.255",
"ipaddr": "10.20.100.248",
"name": "eth0",
"netmask": "255.255.255.0",
"speed": "unknown"
},
{ "broadcast": "255.255.255.255",
"ipaddr": "127.0.0.1",
"name": "lo",
"netmask": "255.0.0.0",
"speed": "locallink"
}
],
"total_memory": 3833,
"vertica": {
"arch": "x86_64",
"brand": "vertica",
"release": "20140716",
"version": "9.0.0"
}
},
{ "cpu_info": {
"cpu_type": "Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
"number_of_cpus": 2
},
"host_id": "10.20.100.249",
"hostname": "v_vmart_node0003.example.com",
"max_user_proc": "3833",
"nics": [
{ "broadcast": "10.20.100.255",
"ipaddr": "10.20.100.249",
"name": "eth0",
"netmask": "255.255.255.0",
"speed": "unknown"
}
"netmask": "255.255.255.0",
"speed": "unknown"
},
{
"broadcast": "255.255.255.255",
"ipaddr": "127.0.0.1",
"name": "lo",
"netmask": "255.0.0.0",
"speed": "locallink"
}
],
"total_memory": 3833,
"vertica": {
"arch": "x86_64",
"brand": "vertica",
"release": "20140716",
"version": "9.0.0"
}
],
"href": "/hosts",
"links": [
"/:hostid"
],
"mime-type": "application/vertica.hosts.json-v2"}
GET hosts/:hostid

Returns hardware, software, and network details about the host identified by :host_id. You can find :host_id for each host using GET hosts.

Resource URL

https://<NODE>:5444/hosts/:hostid

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

```
| GET     | https://<NODE>:5444/hosts/10.20.100.247 |
```

Response:

```
{
  "body": {
    "cpu_info": {
      "cpu_type": " Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz",
      "number_of_cpus": 2
    },
    "hostname": "v_vmart_node0001.example.com",
    "max_user_proc": "3833",
    "nics": [
      {
        "broadcast": "10.20.100.255",
        "ipaddr": "10.20.100.247",
        "name": "eth0",
        "netmask": "255.255.255.0",
        "speed": "unknown"
      },
      {
        "broadcast": "255.255.255.255",
        "ipaddr": "127.0.0.1",
        ...
      }
    ]
  }
}``
```
Jobs

You can use these API calls to get information on your database's jobs.

<table>
<thead>
<tr>
<th>GET jobs</th>
<th>Returns a list of jobs the agent is tracking, along with their current status and exit codes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET jobs/:id</td>
<td>Returns the details (the saved output) for a specific job.</td>
</tr>
</tbody>
</table>
GET jobs

Returns a list of jobs being tracked by the agent and job details.

Jobs always start immediately. The is_running field is a Boolean value. If is_running is false, then the job is complete.

The exit_code details the status of the job. The exit_code is different for certain types of jobs:

- For Backup jobs:
  - 0 indicates success.
  - Any other number indicates a failure.

- For all other jobs:
  - -9 indicates success.
  - Any other number indicates a failure.

You can see details about failures in /opt/vertica/log/agentStdMsg.log.

Resource URL

https://<NODE>:5444/jobs

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET   | https://<NODE>:5444/jobs |
Response:

```json
{
  "body": [
    {
      "exit_code": 0,
      "id": "CreateBackup-VMart-1405012447.75",
      "is_running": false,
      "status": "unused",
      "ts": "1405012461.18"
    },
    {
      "exit_code": 1,
      "id": "CreateBackup-VMart-1405012454.88",
      "is_running": false,
      "status": "unused",
      "ts": "1405012455.18"
    }
  ],
  "href": "/jobs",
  "links": [
    "/:jobid"
  ],
  "mime-type": "application/vertica.jobs.json-v2"
}
```
GET jobs/:id

 Gets the details for a specific job with the provided :id. You can determine the list of job :ids using GET jobs.

 Details for a specific job are the same as the details provided for all jobs by GET jobs.

 Note: You must URL encode the :id as some IDs may contain spaces or other special characters.

 Resource URL

 https://<NODE>:5444/jobs/:id

 Authentication

 Requires a VerticaAPIKey in the request header.
 The API key must have restricted level security or higher.

 Parameters

 None.

 Example Request

| GET | https://<NODE>:5444/jobs/CreateBackup-VMart-1405012454.88 |

 Licenses

 You can use these API calls to manage licenses for your database.

| POST licenses | Uploads and applies a new license to this cluster. |
| GET licenses  | Returns the license field that databases created on this cluster use. |
**POST licenses**

Uploads and applies a license file to this cluster.

You must provide the license file as an HTTP POST form upload, identified by the name `license`. For example, you can use cURL:

```
curl -k --request POST -H "VerticaApiKey:ValidAPIKey" \
https://v_vmart_node0001:5444/licenses --form "license=@vlicense.dat"
```

**Resource URL**

https://<NODE>:5444/licenses

**Authentication**

Requires a VerticaAPIKey in the request header.

The API key must have admin level security.

**Parameters**

None.

**Example Request**

```
POST https://<NODE>:5444/licenses
```

**Response:**

There is no HTTP body response for successful uploads. A successful upload returns an HTTP 200/OK header.
GET licenses

Returns any license files that are used by this cluster when creating databases. License files must reside in /opt/vertica/config/share.

Resource URL

https://<NODE>:5444/licenses

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET         | https://<NODE>:5444/licenses |

Response:

```json
{
  "body": [
    {
      "comment": "Vertica license is valid",
      "end": "Perpetual",
      "grace": "0",
      "size": "1TB CE Nodes 3",
      "start": "2011-11-22",
      "status": true,
      "vendor": "Vertica Community Edition"
    }
  ],
  "href": "/license",
  "links": [],
  "mime-type": "application/vertica.license.json-v2"
}
```
## Nodes

You can use these API calls to retrieve information on the nodes in your cluster.

<table>
<thead>
<tr>
<th>API Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET nodes</td>
<td>Returns a list of nodes in this cluster.</td>
</tr>
<tr>
<td>GET nodes/:nodeid</td>
<td>Returns details for a specific node in this cluster.</td>
</tr>
</tbody>
</table>
GET nodes

Returns a list of nodes associated with this cluster.

Resource URL

https://<NODE>:5444/nodes

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| GET | https://<NODE>:5444/nodes |

Response:

```json
{
  "body": [
    "node0001",
    "node0002",
    "node0003",
    "v_testdb_node0001",
    "v_testdb_node0002",
    "v_testdb_node0003",
    "v_vmart_node0001",
    "v_vmart_node0002",
    "v_vmart_node0003"
  ],
  "href": "/nodes",
  "links": [
    "/:nodeid"
  ],
  "mime-type": "application/vertica.nodes.json-v2"
}
```
GET nodes/:nodeid

Returns details about the node identified by :node_id. You can find the :node_id for each node using GET nodes.

In the body field, the following information is detailed in comma-separated format:

- Node Name
- Host Address
- Catalog Directory
- Data Directory

Resource URL

```
https://<NODE>:5444/nodes/:node_id
```

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

```
GET
https://<NODE>:5444/nodes/v_vmart_node0001
```

Response:

```
{
   "body": [
      "v_vmart_node0001",
      "10.20.100.247,/home/dbadmin,/home/dbadmin"
   ],
   "href": "/nodes/v_vmart_node0001",
   "links": []
}
```
Webhooks

You can use these API calls to obtain information on, create, or delete webhooks.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET webhooks</td>
<td>Returns a list of active webhooks.</td>
</tr>
<tr>
<td>POST webhooks/subscribe</td>
<td>Creates a new webhook.</td>
</tr>
<tr>
<td>DELETE webhooks/:subscriber_id</td>
<td>Deletes an existing webhook.</td>
</tr>
</tbody>
</table>
GET webhooks

Returns a list of active webhooks for this cluster.

Resource URL

https://<NODE>:5444/webhooks

Authentication

Requires a VerticaAPIKey in the request header.

The API key must have restricted level security or higher.

Parameters

None.

Example Request

```
GET [GET]
https://<NODE>:5444/webhooks
```

Response:

```
{
   "body": [
      {
         "host": "192.168.232.1",
         "id": "79c1c8a18be02804b3d2f48ea6462909",
         "port": 80,
         "timestamp": "2014-07-20 22:54:09.829642",
         "url": "/gettest.htm"
      },
      {
         "host": "192.168.232.1",
         "id": "9c32cb0f3d2f9a7cb10835f1732fd4a7",
         "port": 80,
         "timestamp": "2014-07-20 22:54:09.829707",
         "url": "/getwebhook.php"
      }
   ],
   "href": "/webhooks",
   "links": [
      "/subscribe",
      "/:subscriber_id"
   ]
}
```
},
    "mime-type": "application/vertica.webhooks.json-v2"
}
POST webhooks/subscribe

Creates a subscription for a webhook.

Resource URL

https://<NODE>:5444/webhooks/subscribe

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

<table>
<thead>
<tr>
<th>url</th>
<th>A URL to an application that accepts JSON messages from this cluster.</th>
</tr>
</thead>
</table>

Example Request


Response:

The response is not JSON encoded. The only text response is the ID of the webhook subscription. Additionally, an HTTP 200/OK header indicates success.

79c1c8a18be02804b3d2f48ea6462909
DELETE webhooks/:subscriber_id

Deletes the webhook identified by :subscriber_id. The :subscriber_id is the value of the id field that the GET webhooks command returns.

Resource URL

```markdown
https://<NODE>:5444/webhooks/:subscriber_id
```

Authentication

Requires a VerticaAPIKey in the request header.
The API key must have restricted level security or higher.

Parameters

None.

Example Request

| DELETE | https://<NODE>:5444/webhooks/79c1c8a18be02804b3d2f48ea6462909 |

Response:

There is no HTTP body response for successful deletes. A successful delete returns an HTTP 200/OK header.

Rest APIs for the Management Console

These API calls interact with Management Console nodes.

Alerts

| GET alerts | Returns alerts for the current user. |
Time Information

| GET mcTimeInfo | Returns the current time for the MC server and the timezone of the location where the MC server is located. |

MC-User-ApIKey

The MC-User-ApIKey is a user-specific key used with Management Console. Users must have an MC-User-ApIKey to interact with MC using the Rest API. All users with roles other than None automatically receive an MC-User-ApIKey.

This key grants users the same rights through the API that they have available through their MC roles. To interact with the MC, users pass the key in the request header for the API.

View the MC-User-ApIKey

If you are the database administrator, you can view the MC-User-ApIKey for all users. Individual users can view their own keys.

1. Connect to MC and go to MC Settings > User Management.
2. Select the user to view and click Edit. The user's key appears in the User API Key field.

GET alerts

Returns a list of MC alerts, their current status, and database properties.

Resource URL

https://<MC_NODE>:5450/webui/api/alerts
Authentication

Requires an **MC-User-Apikey** in the request header.

Filter Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>types</strong></td>
<td>The type of alert to retrieve. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• info</td>
</tr>
<tr>
<td></td>
<td>• notice</td>
</tr>
<tr>
<td></td>
<td>• warning</td>
</tr>
<tr>
<td></td>
<td>• error</td>
</tr>
<tr>
<td></td>
<td>• critical</td>
</tr>
<tr>
<td></td>
<td>• alert</td>
</tr>
<tr>
<td></td>
<td>• emergency</td>
</tr>
<tr>
<td><strong>category</strong></td>
<td>For information, see <strong>Thresholds Category Filter</strong>.</td>
</tr>
<tr>
<td><strong>db_name</strong></td>
<td>For information, see <strong>Database Name Category Filter</strong>.</td>
</tr>
<tr>
<td><strong>limit</strong></td>
<td>The maximum number of alerts to retrieve. If the limit is lower than the number of existing alerts, Vertica retrieves the most recent alerts. Used with the type parameter, Vertica retrieves up to the limit for each type. For example, for a limit of 5 and types of critical and emergency, you could receive up to ten total alerts.</td>
</tr>
<tr>
<td><strong>time_from</strong></td>
<td>The timestamp start point from which to retrieve alerts. You can use this parameter in combination with the <strong>time_to</strong> parameter to retrieve alerts for a specific time range. Values must be passed in the following format: <code>yyyy-MM-ddTHH:mm</code>. If you provide only the <strong>time_from</strong> parameter, and omit the <strong>time_to</strong> parameter, the response contains all alerts generated from the <strong>time_from</strong> parameter to the current time.</td>
</tr>
<tr>
<td><strong>time_to</strong></td>
<td>The timestamp end point from which to retrieve alerts. You can use this</td>
</tr>
</tbody>
</table>
parameter in combination with the time_from parameter to retrieve alerts for a specific time range. Values must be passed in the following format: yyyy-MM-ddTHH:mm.

If you provide only the time_to parameter, and omit the time_from parameter, the response contains all alerts generated from the earliest possible time to the time passed in time_to.

Example Request

| GET | https://<MC_NODE>:5450/webui/api/alerts |

Request Alerts Using cURL

This example shows how you can request alerts using cURL. In this example, the limit parameter is set to '2' and the types parameters is set to info and notice:


Response:

```
[
  {
    "alerts":[
      {
        "id":5502,
        "markedRead":false,
        "eventTypeCode":0,
        "create_time":"2016-02-02 05:12:10.0",
        "updated_time":"2016-02-02 15:50:20.511",
        "severity":"warning",
        "status":1,
        "nodeName":"v_vmart_node0001",
        "databaseName":"VMart",
        "databaseId":1,
        "clusterName":"1449695416208_cluster",
        "description":"Warning: Low disk space detected (73% in use)",
        "summary":"Low Disk Space",
        "internal":false,
        "count":3830
      },
      {
        "id":5501,
        "markedRead":false,
        "eventTypeCode":2,
        "create_time":"2016-02-02 05:12:02.31",
        "updated_time":"2016-02-02 05:12:02.31",
      }
    ]
  }
]```
Request Alerts Within a Time Range

These examples show various ways in which you can request the same alert as in the preceding example, but within specified time ranges.

Request the alert within a specific time range, using the `time_from` and `time_to` parameters:

```bash
```

Request the alert from a specific start time to the present using the `time_from` parameter:

```bash
```

Request the alert to a specific end point using the `time_to` parameter. When you use the `time_to` parameter without the `time_from` parameter, the `time_from` parameter defaults to the oldest alerts your MC contains:

```bash
```

GET mcTimeInfo

Returns the current time for the MC server and the timezone where the MC server is located.

Resource URL

https://<MC_NODE>:5450/webui/api/mcTimeInfo
Authentication

Requires an **MC-User-Apikey** in the request header.

Parameters

None.

Example Request

| GET | https://<MC_NODE>:5450/webui/api/mcTimeInfo |

This example shows how you can request MC time information using cURL:

```bash
```

Response:

```
{"mc_current_time":"Tue, 2000-01-01 01:02:03 -0500","mc_timezone":"US/Eastern"}
```

Thresholds Category Filter

Returns a list of alerts related to threshold settings in MC.

Resource URL

https://<MC_NODE>:5450/webui/api/alerts?category=thresholds

Authentication

Requires an **MC-User-Apikey** in the request header.
Example Request

| GET | https://<MC_NODE>:5450/webui/api/alerts?category=thresholds |

This example shows how you can request alerts on thresholds using cURL:


Response:

```
[  
  {  
    "alerts":[  
      {  
        "id":33,  
        "markedRead":false,  
        "eventTypeCode":2,  
        "create_time":"2015-11-10 10:28:41.332",  
        "updated_time":"2015-11-10 10:28:41.332",  
        "severity":"warning",  
        "status":1,  
        "databaseName":"mydb",  
        "databaseId":1,  
        "clusterName":"1446668057043_cluster",  
        "description":"Database: mydb Lower than threshold Node Disk I/O 10 % v_mydb_node0002 ;1.6% v_mydb_node0002 ;1.4% v_mydb_node0002 ;2.3% v_mydb_node0002 ;1.13% v_mydb_node0002 ;1.39% v_mydb_node0002 ;1.79% 
          
          
          
          "summary":"Threshold : Node Disk I/O < 10 %",  
        "internal":false,  
        "count":1  
      },  
      {  
        "id":32,  
        "markedRead":false,  
        "eventTypeCode":2,  
        "create_time":"2015-11-10 10:28:40.975",  
        "updated_time":"2015-11-10 10:28:40.975",  
        "severity":"warning",  
        "status":1,  
        "databaseName":"mydb",  
        "databaseId":1,  
        "clusterName":"1446668057043_cluster",  
        "description":"Database: mydb Lower than threshold Node Memory 10 % v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 ;5.47% v_mydb_node0002 
          
          
          
          
          "summary":"Threshold : Node Memory < 10 %",  
        "internal":false,  
        "count":1  
      },  
      {  
        "id":31,  
        "markedRead":false,  
        "eventTypeCode":2,  
        "create_time":"2015-11-10 10:28:40.531",  
        "updated_time":"2015-11-10 10:28:40.531",  
        "severity":"warning",  
        "status":1,  
        "databaseName":"mydb",  
        "databaseId":1,  
        "clusterName":"1446668057043_cluster",  
        "description":"Database: mydb Lower than threshold Node Memory 10 % v_mydb_node0002 ;4.53% v_mydb_node0002 ;4.53% v_mydb_node0002 ;4.53% v_mydb_node0002 ;4.53% v_mydb_node0002 ;4.53% v_mydb_node0002 
          
          
          
          "summary":"Threshold : Node Memory < 10 %",  
        "internal":false,  
        "count":1  
      }  
  ]  
]  
```
See Also

- **Combining Sub-Category Filters with Category Filters**

**Database Name Category Filter**

Returns a list of MC alerts for a specific database.

**Resource URL**

https://<MC_NODE>:5450/webui/api/alerts?db_name=<database_name>
Authentication

Requires an MC-User-Apkey in the request header.

Example Request

```
GET https://<MC_NODE>:5450/webui/api/alerts?db_name="mydb"
```

This example shows how you can view alerts on a specific database using cURL:

```
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 9,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 15:10:53.391",
        "updated_time": "2015-11-05 15:10:53.391",
        "severity": "notice",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": "Workload analyzed successfully",
        "summary": "Analyze Workload operation has succeeded on Database",
        "internal": false,
        "count": 1
      },
      {
        "id": 8,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 15:10:31.16",
        "updated_time": "2015-11-05 15:10:31.16",
        "severity": "notice",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": "Workload analyzed successfully",
        "summary": "Analyze Workload operation has succeeded on Database",
        "internal": false,
        "count": 1
      },
      {
        "id": 7,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 15:10:06.26",
        "updated_time": "2015-11-05 15:10:06.26",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 1,
        "clusterName": "1446668057043_cluster",
        "description": "Workload analyzed successfully",
        "summary": "Analyze Workload operation has succeeded on Database",
        "internal": false,
        "count": 1
      }
    ]
  }
]
```
"summary":"Analyze Workload operation has succeeded on Database",
"internal":false,
"count":1
},
{
"id":3,
"markedRead":false,
"eventTypeCode":2,
"create_time":"2015-11-04 15:14:32.806",
"updated_time":"2015-11-04 15:14:32.806",
"severity":"info",
"status":1,
"hostIp":"10.20.100.64",
"nodeName":"v_mydb_node0003",
"databaseName":"mydb",
"databaseId":1,
"clusterName":"144668057043_cluster",
"description":"Agent status is UP on IP 127.0.0.1",
"summary":"Agent status is UP on IP 127.0.0.1",
"internal":false,
"count":1
},
{
"id":2,
"markedRead":false,
"eventTypeCode":2,
"create_time":"2015-11-04 15:14:32.541",
"updated_time":"2015-11-04 15:14:32.541",
"severity":"info",
"status":1,
"hostIp":"10.20.100.63",
"nodeName":"v_mydb_node0002",
"databaseName":"mydb",
"databaseId":1,
"clusterName":"144668057043_cluster",
"description":"Agent status is UP on IP 127.0.0.1",
"summary":"Agent status is UP on IP 127.0.0.1",
"internal":false,
"count":1
},
{
"id":1,
"markedRead":false,
"eventTypeCode":2,
"create_time":"2015-11-04 15:14:32.364",
"updated_time":"2015-11-04 15:14:32.364",
"severity":"info",
"status":1,
"hostIp":"10.20.100.62",
"nodeName":"v_mydb_node0001",
"databaseName":"mydb",
"databaseId":1,
"clusterName":"144668057043_cluster",
"description":"Agent status is UP on IP 127.0.0.1",
"summary":"Agent status is UP on IP 127.0.0.1",
"internal":false,
"count":1
}]
"total_alerts":9,
Combining Sub-Category Filters with Category Filters

You can combine category filters with sub-category filters, to obtain alert messages for specific thresholds you set in MC. You can also use sub-category filters to obtain information about alerts on specific resource pools in your database.

Sub-Category Filters

You can use the following sub-category filters with the category filters. Sub-category filters are case sensitive and must be lowercase.

<table>
<thead>
<tr>
<th>Sub-Category Filter</th>
<th>Alerts Related to Threshold Value Set For:</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold_node_cpu</td>
<td>Node CPU</td>
</tr>
<tr>
<td>threshold_node_memory</td>
<td>Node Memory</td>
</tr>
<tr>
<td>threshold_node_disk_usage</td>
<td>Node Disk Usage</td>
</tr>
<tr>
<td>threshold_node_diskio</td>
<td>Node Disk I/O</td>
</tr>
<tr>
<td>threshold_node_cpuio</td>
<td>Node CPU I/O Wait</td>
</tr>
<tr>
<td>threshold_node_rebootrate</td>
<td>Node Reboot Rate</td>
</tr>
<tr>
<td>threshold_netio</td>
<td>Network I/O Error</td>
</tr>
<tr>
<td>threshold_query_queued</td>
<td>Queued Query Number</td>
</tr>
<tr>
<td>threshold_query_failed</td>
<td>Failed Query Number</td>
</tr>
<tr>
<td>threshold_query_spilled</td>
<td>Spilled Query Number</td>
</tr>
<tr>
<td>threshold_query_retried</td>
<td>Retried Query Number</td>
</tr>
<tr>
<td>threshold_query_runtime</td>
<td>Query Running Time</td>
</tr>
</tbody>
</table>
Resource Pool-Specific Sub-Category Filters

To retrieve alerts for a specific resource pool, you can use sub-category filters in combination with the following category filters:

- thresholds
- rp_name

If you use these sub-category filters without the rp_name filter, the query retrieves alerts for all resource pools in your database.

<table>
<thead>
<tr>
<th>Sub-Category Filter</th>
<th>Alerts Related to Threshold Value Set For:</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold_rp_query_max_time</td>
<td>Queries reaching the maximum allowed execution time.</td>
</tr>
<tr>
<td>threshold_rp_query_resource_reject</td>
<td>The number of queries with resource rejections.</td>
</tr>
<tr>
<td>threshold_rp_query_queue_time</td>
<td>The number of queries that ended because of queue time exceeding a limit.</td>
</tr>
<tr>
<td>threshold_rp_query_run_time</td>
<td>The number of queries that ended because of run time exceeding a limit.</td>
</tr>
<tr>
<td>threshold_rp_memory</td>
<td>The minimum allowed resource pool size.</td>
</tr>
<tr>
<td>threshold_rp_max_memory</td>
<td>The maximum allowed resource pool size.</td>
</tr>
</tbody>
</table>

Authentication

Requires an MC-User-Apikey in the request header.

Example Request

```
```
Combine the Thresholds Category Filter with a Sub-Category Filter

This example shows how you can request alerts using cURL with the thresholds category filter and a sub-category filter. You apply the following filters:

- thresholds
- threshold_node_cpu

```
```

Response:

```
[
  {
    "alerts": [
      {
        "id": 11749,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 11:04:43.997",
        "updated_time": "2015-11-05 11:04:43.997",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb",
        "databaseId": 105,
        "clusterName": "1443122180317_cluster",
        "description": "Database: mydb Lower than threshold Node CPU < 10 % v_mydb_node0002 ; 1.03% v_mydb_node0003 ; 0.9% v_mydb_node0001 ; 1.36%%",
        "summary": "Threshold: Node CPU < 10 %",
        "internal": false,
        "count": 1
      },
      {
        "id": 11744,
        "markedRead": false,
        "eventTypeCode": 2,
        "create_time": "2015-11-05 10:59:46.107",
        "updated_time": "2015-11-05 10:59:46.107",
        "severity": "warning",
        "status": 1,
        "databaseName": "mydb2",
        "databaseId": 106,
        "clusterName": "1443552354071_cluster",
        "description": "Database: mydb2 Lower than threshold Node CPU < 10 % v_mydb2_node0002 ; 0.83% v_mydb2_node0001 ; 1.14%",
        "summary": "Threshold: Node CPU < 10 %",
        "internal": false,
        "count": 1
      }
    ]
  }
]
Request an Alert On a Specific Resource Pool

This example shows how you can request alerts using cURL on a specific resource pool. The name of the resource pool is `resourcepool1`. You apply the following filters:

- `thresholds`
- `rp_name`
- `threshold_rp_query_run_time`

```
```

Response:

```
[{
  "alerts": [ 
  {
    "id":6525,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-05 14:25:36.797",
    "updated_time":"2015-11-05 14:25:36.797",
    "severity":"warning",
    "status":1,
    "databaseName":"mydb",
    "databaseId":106,
    "clusterName":"1443552354071_cluster",
    "description":"Resource Pool: resourcepool1 Threshold Name: Ended Query with Run Time Exceeding Limit Time Interval: 14:20:36 to 14:25:36 Threshold Value: 0 min(s) Actual Value: 2186 query(s)",
    "summary":"Resource Pool: resourcepool1; Threshold : Ended Query with Run Time Exceeding Limit > 0 min(s)",
    "internal":false,
    "count":1
  },
  {
    "id":6517,
    "markedRead":false,
    "eventTypeCode":2,
    "create_time":"2015-11-05 14:20:39.541",
    "updated_time":"2015-11-05 14:20:39.541",
  }
]
```
Severity: warning
Status: 1
Database Name: mydb
Database ID: 106
Cluster Name: 144352354071_cluster
Description: Resource Pool: resourcepool1 Threshold Name: Ended Query with Run Time Exceeding Limit Time Interval: 14:15:39 to 14:20:39 Threshold Value: 0 min(s) Actual Value: 2259 query(s)
Summary: Resource Pool: resourcepool1; Threshold: Ended Query with Run Time Exceeding Limit > 0 min(s)
Internal: false
Count: 1
Total alerts: 14
Request query: category=thresholds&subcategory=threshold_rp_query_run_time&rp_name=resourcepool1
Request time: 2015-11-05 11:07:43.988
Using Eon Mode Beta

Vertica 9.0.0 allows you to operate your database in Eon Mode Beta. Doing so separates the computational processes from the storage layer of your database, thereby enabling rapid scaling of computational resources to accommodate variable demand workloads. Initial deployment of Eon Mode Beta is limited to Amazon Web Services.

Important: This functionality is currently in Beta. Vertica provides this functionality for testing and evaluation purposes, but does not support running Eon Mode Beta in a production environment.

Vertica will not provide technical support for Eon Mode Beta users, but you can use the VerticaBeta Forum to ask and answer questions about operating your Vertica database in Eon Mode Beta.

To learn more about trying Eon Mode Beta, see https://www.vertica.com/eonmodebeta/.

While it is possible to upgrade a database running in Eon Mode Beta, Vertica does not support this functionality while the feature is still in beta. This includes:

• Upgrading a 9.0.0 Eon Mode Beta database to later versions of Vertica.
• Reviving a 9.0.0 Eon Mode Beta database using a later version of Vertica.
Eon Database Overview

Vertica Eon mode separates the computational processes from the storage layer of your database. This separation gives you the ability to store your data in a single location and elastically vary the number of nodes connected to that location according to your computational needs. You can adjust the size of your cluster without interrupting analytic workloads, adding or removing nodes as the volume of work changes.

Eon Terminology

The following table defines the new concepts introduced in Eon Mode:

<table>
<thead>
<tr>
<th>Vertica Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eon Mode</td>
<td>Vertica mode that decouples local node compute and storage.</td>
</tr>
</tbody>
</table>
### Vertica Concept | Definition
--- | ---
**Enterprise Mode** | Vertica mode that combines local node compute and storage. Previously, this was the only mode in Vertica.
**Communal Storage** | A storage location shared amongst a database. For example, Amazon S3. This type of storage is required when running in Eon mode.
**Node Storage** | A storage location on each node in the database.
**Depot** | A local copy of data associated with shards that a node subscribes to.
**Shards** | Shards are a subset of the metadata and associated data from your communal storage location. At the time of database creation you define the number of shards you want to have in your database.
There are two different types of shards: replica and segment. The database defines the number of segment shards at database creation time. Segment shards determine the maximum number of nodes that will source storage data in a query. Each shard contains one replica shard. Replica shards contain all metadata and data expected to be fully replicated. This includes unsegmented projections.
**Node Subscriptions** | A node subscribed to a shard has access to the metadata for the shard and relevant data cached. Subscriptions have states that declare what information the node has and how it could participate in queries or other operations.
Multiple nodes may subscribe to the same shard. A node may subscribe to more than one shard.
**Elastic Throughput Scaling** | The process of storing redundant copies of existing shards on new nodes. You can plan queries on a subset of the database nodes for increased throughput.

### Architecture and Concepts
Eon Mode introduces a different architecture and new terminology to Vertica. The following topics explain these changes in detail.
Eon Mode Architecture

In Eon mode, the physical architecture is designed to separate the storage layer from the computation layer. This is different from Enterprise mode, which couples the storage and compute layers.

Depot

The depot is a cache-like component that creates a local copy of data associated with a shard that a node subscribes to. Each node in the cluster subscribes to a different shard or shards of the communal storage. The depot improves query performance by preventing some queries from making the long round trip to communal storage.

Data Storage

In Eon mode, COPY statements write to Read Optimized Store (ROS) files in a node's depot. The COPY statement segments, sorts, and compresses for high optimization. Before the statement commits, Vertica ships the ROS files to communal storage. Eon mode does not use the Write Optimized Storage (WOS).

The following diagram illustrates how data flows during a COPY statement.
Communal Storage

Instead of storing data locally, Eon Mode uses a singular communal storage location for all data. Communal storage is the database's centralized storage location, shared amongst the database or multiple nodes. This can be on a public cloud deployment like Amazon S3.

You can connect nodes to communal storage according to computing need. The separation of the storage layer in Eon mode allows you to elastically scale your cluster more easily, without data redistribution interrupting your analytic workload.
Communal storage locations are listed in the STORAGE_LOCATIONS system table with a SHARING_TYPE of COMMUNAL.

**Shards and Subscriptions**

In Eon mode, you define the number of shards at the time you create your database. Shards are segments of the data in your communal storage location. The shards in your communal storage location are similar to a collection of segmented projections in an Enterprise mode database. Each node in your database subscribes to a subset of the shards in the communal storage location. This ensures that if a node goes down or a node is being used in another query, all the data is available on the remaining nodes.
Expanding Your Cluster

This diagram shows the process of expanding a 3 node 3 shard database to a 6 node 3 shard database. The number of shards stays the same and the additional nodes gain subscriptions to the shards. When nodes join the Vertica cluster they immediately subscribe to shards and begin the process of populating their depots.

Depot

The depot provides the node with a local copy of data stored on the communal storage. Each node in an Eon database has its own depot. By default, the depot is set to be 80% of the disk space on the node. Vertica recommends that you use instance volumes for your depots. This intermediate layer of data storage provides a faster copy of the data that is local to the node. Data that is frequently used in your queries takes priority in your depot. If the data for a query is not in the depot, then Vertica might need to read data from communal storage.

Loading Data

In Eon mode, COPY statements write to Read Optimized Store (ROS) files in a node's depot. The COPY statement segments, sorts, and compresses for high optimization. Before the statement commits, Vertica ships the ROS files to communal storage. Eon Mode does not use the Write Optimized Storage (WOS).
As the data is being loaded two processes are running in parallel across the database: ROS files are being written locally and stored in the depot, and the ROS files are uploaded to the communal storage.

The following animation shows the flow of data during a copy statement.

![Diagram of data flow]

Querying Data

Vertica first tries to use data in the depot to resolve a query. When the data in the depot cannot resolve the query, the query reads from the communal storage. You could see poor query performance when a substantial number of your queries read from the communal storage. If this is the case, then you should consider re-sizing your depot or use depot system tables to get a better idea of what is causing the issue.

Elasticity in Vertica in Eon Mode Beta

Elasticity, or Elastic Throughput Scaling (ETS), allows you to increase throughput of the existing cluster by running queries on a subset of a full cluster. This subset subscribes to all necessary segments. For example, you have 20 nodes, 10 segments, with 2 nodes subscribed to each segment. This means you can run a query on 10 nodes instead of all 20, running 2 queries at the same without stepping on each other, getting increased throughput. In this example, the
10 nodes subscribed to the full segment set are Participating Subscriptions, see Participating Subscriptions For a Session.

ETS also allows you to temporarily add nodes to your cluster to provide higher query throughput. For example, you may regularly run a 10-node cluster. But when user numbers increase, you may need to run 20 nodes to meet the increased demand.

Vertica in Eon Mode Beta accomplishes this using the following:

- **REBALANCE_SHARDS**
- Participating subscriptions for a session
- Constraints

## REBALANCE_SHARDS

When you run REBALANCE_SHARDS, Vertica in Eon Mode Beta rebalances shard assignments across the nodes in a cluster. Run REBALANCE_SHARDS after you modify your cluster using ALTER NODE. The output should have the following properties for a specific number of nodes and shards, as well as a specific fault tolerance (K-Safety).

- If a shard has Participating Subscribers, it must have at least one more subscription than the fault tolerance level.
- Each participating subscription size greater than the number of shards must contain a full set of subscriptions from its nodes to all the shards.
- All nodes must be subscribed to the replica shard. Replica shards contain all metadata and data for storage that is expected to be fully replicated. Some of those data are unsegmented projection data and DFS files.
- Short-lived nodes and execute nodes must not have subscriptions.

Check the **NODE_SUBSCRIPTIONS** table to verify that running REBALANCE_SHARDS provides the expected output, for example:

```sql
=> SELECT node_name, shard_name, subscription_state, is_primary FROM node_subscriptions order by node_name, shard_name;
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>shard_name</th>
<th>subscription_state</th>
<th>is_primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>node01</td>
<td>replica</td>
<td>ACTIVE</td>
<td>t</td>
</tr>
<tr>
<td>node01</td>
<td>segment0001</td>
<td>ACTIVE</td>
<td>t</td>
</tr>
<tr>
<td>node02</td>
<td>replica</td>
<td>ACTIVE</td>
<td>f</td>
</tr>
<tr>
<td>node02</td>
<td>segment0002</td>
<td>ACTIVE</td>
<td>t</td>
</tr>
<tr>
<td>node03</td>
<td>replica</td>
<td>ACTIVE</td>
<td>f</td>
</tr>
</tbody>
</table>
In the NODE_SUBSCRIPTIONS table the is_primary column indicates whether the node is currently the primary subscriber.

After running REBALANCE_SHARDS check the subscription status by querying the SESSION_SUBSCRIPTIONS table, for example:

```sql
=> SELECT * from session_subscriptions where is_participating='t' order by node_name, shard_name;
```

<table>
<thead>
<tr>
<th>node_oid</th>
<th>node_name</th>
<th>shard_oid</th>
<th>shard_name</th>
<th>is_participating</th>
</tr>
</thead>
<tbody>
<tr>
<td>45035996273704978</td>
<td>node01</td>
<td>45035996273704980</td>
<td>replica</td>
<td>t</td>
</tr>
<tr>
<td>45035996273704978</td>
<td>node01</td>
<td>45035996273704992</td>
<td>segment003</td>
<td>t</td>
</tr>
<tr>
<td>45035996273705124</td>
<td>node02</td>
<td>45035996273704980</td>
<td>replica</td>
<td>t</td>
</tr>
<tr>
<td>45035996273705124</td>
<td>node02</td>
<td>45035996273704988</td>
<td>segment002</td>
<td>t</td>
</tr>
<tr>
<td>45035996273946332</td>
<td>node04</td>
<td>45035996273704984</td>
<td>segment001</td>
<td>t</td>
</tr>
</tbody>
</table>

**Participating Subscriptions For a Session**

When running, sessions select a set of subscriptions to provide data to segmented or replicated projections. Participating subscriptions remain active for the session unless the node goes down or a subscription changes in such a way as to invalidate the existing selection.

Sessions select participating subscriptions using the following guidelines:

- Only nodes that are UP and have ACTIVE subscriptions can provide a subscription.
- Participating nodes must provide a balanced number of shards.
- The initiator node should be included, if possible.
- Nodes from the initiator’s participating subscription spill out to other nodes only when necessary.
- A node that serves a segment must also serve the replica shard.

Use the SESSION_SUBSCRIPTIONS table to determine the active nodes with the is_participating column. The is_participating column indicates whether a subscription is participating in a session. The following example shows a cluster with 8 nodes, 4 shards, and a database with 2 fault groups connecting to node01:
Constraints

Elastic Throughput Scaling is disabled for the following DML statements:

- **INSERT**
- **DELETE**
- **COPY** Direct
- **Trickle Loading Data**
- **MERGE**

ETS is disabled for the above DML statements with the exception of some cases of COPY:

- You can only run COPY ON NODE and COPY ON ANY NODE on the nodes with participating subscriptions. See **COPY Parameters** for more information.

- Vertica in Eon Mode Beta supports ETS in SELECT/COPY for external tables created with COPY ... ON ANY NODE or equivalent. For more information see **Using External Tables**.

Peer-to-Peer Metadata Transfer in Eon Mode Beta

In Vertica Eon Mode Beta, the primary data storage is a shared storage between all Vertica database nodes. The data is segmented (partitioned) into shards for ease of data access. To mitigate shared Storage I/O contention, each database node subscribes to shards. When K-Safety is 1 or higher (high availability), each shard has more than one node subscriber. In this
configuration where nodes share shards is called Peer-to-Peer configuration. It creates the metadata redundancy for the secondary data storage on the local nodes.

For the efficient transfer and synchronization of shard metadata between peer nodes, the metadata gets transferred between shard subscriber nodes, this is called Peer to Peer Metadata Transfer. This applies only to metadata. The actual data that is stored on each node depot gets copied from share storage as needed.

Peer-to-peer data transfer fetches data from active neighbor nodes resulting in faster returns on queries. It also ensures that all nodes that subscribe to a shard contain all the metadata for all the objects in the shard.

There are two types of shards:

Segment shard - each shard has a hash range bound. Storage metadata for data in that hash range gets created under this segment shard. For example, all segmented projections create/read metadata in segment shards. Nodes must subscribe to segment shards to get the associated metadata.

Replica shards - do not have any hash bound and only one exists per node. All unsegmented projections create/read metadata in a replica shard. All nodes have metadata associated with replica shard.

Peer-to-peer data transfer occurs in the following modes:

- Online
- Batch

Online

When Vertica creates a storage ID in a shard, the metadata is sent to all the subscribers of that shard.

Batch

When a node comes up it receives metadata from its peers and then transitions to a Passive state. Under PASSIVE state, it fetches data directly from S3 to warm its depot and transitions to ACTIVE state.
Eon Mode Beta Installation and Database Creation

You can install an Eon Mode Beta database using in-browser wizards provided by Vertica Management Console with Provisioning, available through the AWS Marketplace.

If you have many specific AWS configuration needs, you can instead use the command line to install Vertica on AWS resources and create an Eon Mode Beta database.

See Installing Vertica in Eon Mode Beta in the Installing Vertica guide for Eon Mode Beta installation procedures.
Walkthrough: Eon Functionality

The walkthrough demonstrates how to load data, run queries, and add nodes in an Eon Mode Beta database. The walkthrough also includes running a script that simulates concurrent processes running on your database. The data used in the walkthrough is the VMart data set included with the Vertica Analytic Database.

Each section of the walkthrough highlights a specific functional area of an Eon Mode Beta database.

- Loading data – Data is loaded by writing ROS files to the depots on each of the nodes and then sending those ROS files to the Communal Storage.
- Querying data – Run queries and achieve similar query performance to Vertica running in Enterprise mode.
- Adding nodes – Add nodes to your cluster and improve the throughput of your database.
- Running scrutinize – Running scrutinize collects a wide range of information from all the nodes in your cluster.

Before you begin this walkthrough, make sure you have familiarized yourself with the topics in Conceptual Overviews and Terms. Having an understanding of Eon Mode Beta concepts and terms should make this walkthrough a great practical application of your knowledge.

Requirements

For this walkthrough there are a few specific requirements you should follow:

- You must use Amazon Web Services (AWS) and S3.
- Vertica recommends using c3.8xlarge instance types. If this instance type is not available in your region, then a similar instance type should work.
- Vertica recommends using MC with Provisioning to create your database.
- Ensure that all of your nodes are in the same IAM security group.
- Your nodes should use instance storage.
- Your cluster should contain 6 nodes.
Getting Started

The first thing you need to do is install the Eon Mode Betapackage on all the nodes in your cluster.

There are two different ways you can perform the installation:

- Installing Using Management Console with Provisioning
- Installing Vertica in Eon Mode Beta Using the Command Line

Creating an Eon Mode Beta Database

In this walkthrough we want to highlight the elasticity benefits from using Eon mode. To do this we are going to create a database with 3 nodes and 3 shards. That means that the other 3 nodes in your cluster are not included in your create database command.

Important: If you used MC with Provisioning to create your database, skip this step.

Use the following steps to create a 3 node and 3 shard Eon Mode Beta database.

1. As the database administrator, create a database using the following arguments:
   - Database Name: --database
   - Host List: --hosts
   - Depot Path: --depot-path
   - Shard Count: --shard-count
   - Communal Storage Location: --communal-storage-location

   ```
   $ /opt/vertica/bin/admintools -t create_db --database <db name> \ 
   --hosts <host list> \ 
   --depot-path <depot path> \ 
   --shard-count <number of shards> \ 
   --communal-storage-location <communal storage location>
   ```
Loading Data

Loading data in an Eon database is the same as loading data in an Enterprise database. Eon databases write directly to ROS files. The ROS files are written directly to the depot and uploaded to the communal storage location in parallel.

To simulate a parallel load across the database we are going to run a shell script. The script takes three parameters: number of concurrent vsql threads, duration, and schema name.

1. First SSH to your database instance as dbadmin, using your .pem key and instance's IP address.

   If you created your database using the CloudFormation Template, your SSH access information is on the CloudFormation Stacks page under your stack's Outputs tab, labeled SSH.

   For example:

   ```
   $ ssh -i example.pem dbadmin@xx.xxx.xxx.xx
   ```

2. Enter your dbadmin password to automatically use for the connection, and set the number of concurrent vsql threads to be 60, the duration to be 300 (5 minutes or 300 seconds), and the schema name to be `concurrent_load`.

   ```
   $ VSQL_PASSWORD=examplepassword /opt/vertica/examples/VMart_Schema/execute_load_in_parallel.sh 60 300 concurrent_load
   ```

   Note that the load may generate some errors. Proceed to steps 3 and 4 to confirm if the load succeeded.

3. To observe the results in system tables, connect to the database.

   ```
   $ vsq1
   ```

4. Run the following queries to view results of the load.

   Workload distribution during the load:

   ```
   => SELECT dqe.node_name, 
   COUNT(*)
   ```
FROM dc_query_executions AS dqe
WHERE dqe.execution_step='Plan'
 AND EXISTS
 (SELECT 1
  FROM dc_requests_issued AS dri
  WHERE dri.transaction_id = dqe.transaction_id
  AND dri.statement_id = dqe.statement_id
  AND dri.request_type = 'LOAD')
GROUP BY node_name
ORDER BY node_name;

The number of successful COPY statements during the load:

=> SELECT COUNT(*)
  FROM dc_requests_issued dri,
       dc_requests_completed AS drc
  WHERE dri.request_type = 'LOAD'
  AND drc.success = 't'
  AND dri.session_id = drc.session_id
  AND dri.request_id = drc.request_id;

Observe the results in Vertica Management Console (MC):

1. Log in. If you created your database using the CloudFormation Template, Vertica MC is already fully deployed. Your MC access information is on the CloudFormation Stacks page under your stack's Outputs tab, labeled Management Console.

   The URL for MC includes the IP address of the host on which MC is installed, followed by the MC port: https://xx.xxx.xxx.xxx:5450/webui

2. Once you have logged into MC, select your database from the Home page to see your database's Overview page. This page gives a summary of your database's activity and system health.

3. Select the Load tab at the bottom of the page to view the Data Load Activity page. The Instance tab displays the status of all the load tasks you have just performed. Note the execution times of these load tasks with 3 nodes in the cluster.

   For this load, MC will display many pages of results. You can click through using the page buttons at the bottom to see the status of your loads. If the load script encountered errors but the load was ultimately successful, some loads may be marked as failed, while the majority should be labeled successful.
4. Now that you have loaded data into the depot, you can also monitor your depot's capacity.

From the Home page, go to Infrastructure > View Storage Allocation.

The Databases Storage tab displays your storage location details, and the Depot Usage by Database tab displays the amount of depot storage now in use across your cluster.

-------------------

Generating and Loading the VMart Data

Before running the script you need to load the VMart data set.

If you are still in vsql, enter \q to quit.

As dbadmin, use the following commands to generate and load the data:

```
$ cd /opt/vertica/examples/VMart_Schema
$ ./vmart_gen
```

Execute the create schema and load data files.
Querying Data

Querying data in an Eon database is the same as in an Enterprise database. All of the Vertica data structures (tables, projections, etc.) work in the same way.

To simulate many parallel queries across your database, we are going to run a shell script. The script takes two parameters: number of concurrent vsql threads and duration.

Include your dbadmin password, set the number of concurrent vsql threads to be 60, and the duration to be 300 (5 minutes or 300 seconds).

```
$ VSQL_PASSWORD=examplepassword /opt/vertica/examples/VMart_Schema/execute_query_in_parallel.sh 60 300
```

Connect to the database to observe the results in system tables.

```
$ vsql
```

Workload distribution during the load:

```
=> SELECT dqe.node_name,
    COUNT(*)
FROM dc_query_executions AS dqe
WHERE dqe.execution_step='Plan'
AND EXISTS
  (SELECT 1
   FROM dc_requests_issued AS dri
   WHERE dri.transaction_id = dqe.transaction_id
   AND dri.statement_id= dqe.statement_id
   AND dri.request_type='LOAD'
  )
GROUP BY node_name
ORDER BY node_name;
```

The number of successful COPY statements during the load:

```
=> SELECT COUNT(*)
FROM dc_requests_issued dri,
    dc_requests_completed AS drc
WHERE dri.request_type='LOAD'
AND drc.success='t'
AND dri.transaction_id=drc.transaction_id
AND dri.request_id=drc.request_id;
```

Observe the results in MC:
You can use the Depot Activity page to monitor the distribution of your depot across the cluster, and whether queries are hitting the depot for optimal performance.

From your database's Overview page, click Activity at the bottom of the page and select Depot Activity Monitoring from the chart menu.

The bar on the left side of the page lists details about depot usage by node.

The User Query Activity by Storage Location chart is a line graph, displaying all queries hitting the depot in green. Any queries hitting communal storage are displayed in red. Hover over lines to view details.

To monitor the status of your queries, select Query Monitoring from the chart menu. The Query Monitoring page displays details about your running and completed queries. At the bottom of the page, four bar charts display information about current queries by node.
Add Your License

If you used the Cloud Formation Template to create your database, then you need to add a valid Vertica license to add nodes to your database. If you already added your license when you created your database, you can skip this step.

$ /opt/vertica/bin/admintools -t upgrade_license_key --install --license examplelicense.key

Adding Nodes

Eon databases use elastic throughput scaling when additional nodes are added to the database. Elastic throughput scaling is the process of storing redundant copies of existing shards on new nodes. This means that queries can be planned on a subset of the database nodes for increased throughput.

To illustrate the benefits of elastic throughput scaling we are going to add the other 3 nodes in our 6 node cluster.

Remember, you should have already installed the Vertica package on the nodes during the Getting Started phase of this Walkthrough. By installing the Vertica package on the nodes before adding the nodes to the cluster, we are expediting the process of adding the nodes to the database.
Using admintools, add the nodes to the database. In the command below, substitute your database name, database password, and comma-separated list of hosts.

```
$ admintools -t db_add_node -d verticadb -p examplepassword -a xx.xx.xx.xx,xx.xx.xx.xx,xx.xx.xx.xx
```

Observe the results in MC:

To see a visualization of your cluster, from the bottom of your database's Overview page select Manage. The Manage page shows an overview of your cluster's node details and health. The page displays standby nodes as gray; once you have successfully added these nodes to your cluster, they will display as green.

Creating Fault Groups

In order to get the extra throughput provided by elastic throughput scaling, you need to create fault groups. This is not adding fault tolerance to your database, but it is providing the database with node groupings.

For more information about creating fault groups, see Creating Fault Groups.

Your fault groups input file should look something like this:

```
group1 group2
  group1 = v_eon_walkthrough_node0001 v_eon_walkthrough_node0002 v_eon_walkthrough_node0003
  group2 = v_eon_walkthrough_node0004 v_eon_walkthrough_node0005 v_eon_walkthrough_node0006
```
Rebalancing Shards

Before you can utilize the extra throughput of elastic throughput scaling, you must rebalance your shards:

```sql
=> SELECT REBALANCE_SHARDS();
REBALANCE_SHARDS
------------------
REBALANCED SHARDS
(1 row)
```

Running the Scripts Again

Re-run the parallel load and query scripts with our new 6 node 3 shard database and measure the performance changes.

Loading Data

Observe the results in system tables.

Set the number of concurrent vsq1 threads to be 60, the duration to be 300 (5 minutes or 300 seconds), and the schema name to be `concurrent_load`.

```bash
$ VSQL_PASSWORD=vertica ./opt/vertica/examples/VMart_Schema/execute_load_in_parallel.sh 60 300 concurrent_load
```

Observe the results in system tables.

Workload distribution during the load:

```sql
=> SELECT dqe.node_name,
    COUNT(*)
FROM dc_query_executions AS dqe
WHERE dqe.execution_step='Plan'
    AND EXISTS
    (SELECT 1
     FROM dc_requests_issued AS dri
     WHERE dri.transaction_id = dqe.transaction_id
     AND dri.request_id= dqe.request_id
     AND dri.request_type= 'LOAD' )
GROUP BY node_name
ORDER BY node_name;
```

The number of successful COPY statements during the load:
SELECT COUNT(*)
FROM dc_requests_issued dri,
     dc_requests_completed AS drc
WHERE dri.request_type='LOAD'
     AND drc.success='t'
     AND dri.session_id=drc.session_id
     AND dri.request_id=drc.request_id;

Observe the results in MC:

From your database's Overview page, select Load > Instance to view load details. Compare the execution time of these loads after adding 3 additional nodes to the cluster.

To check your depot's new capacity, navigate from the Home page to Infrastructure > View Storage Allocation > Depot Usage by Database.

Querying Data

Set the number of concurrent vsql threads to be 60 and the duration to be 300 (5 minutes or 300 seconds).

$ VSQL_PASSWORD=vertica /opt/vertica/examples/VMart_Schema/execute_query_in_parallel.sh 60 300

Observe the results in system tables.

Workload distribution during the load:
The number of successful COPY statements during the load:

```sql
=> SELECT COUNT(*)
    FROM dc_requests_issued dri,
         dc_requests_completed AS drc
    WHERE dri.request_type='LOAD'
      AND drc.success='t'
      AND dri.session_id=drc.session_id
      AND dri.request_id=drc.request_id;
```
Browsing S3 Data Using External Tables

You might want to browse the data in a large data lake that is already stored on S3. Instead of loading data into Vertica, you can read it in place at query time using external tables.

You can create external tables to query Parquet, ORC, text, and delimited data stored in S3 buckets. External tables work the same way in Eon Mode Beta as they do in Enterprise Mode, with one exception: users who are not superusers must have read access to an S3 storage location to run queries. You can create a USER storage location on S3 for this purpose.

The following example illustrates the steps to create a USER storage location, give a user access to it, and create an external table (as that user).

1. Define the AWS credentials associated with the S3 bucket where you will create the storage location:

   ```sql
   => SELECT AWS_SET_CONFIG('aws_id', 'EXAMPLE');
   AWS_SET_CONFIG
   ---------------------
   aws_id
   (1 row)
   => SELECT AWS_SET_CONFIG('aws_secret', 'EXAMPLE/EXAMPLEKEY');
   AWS_SET_CONFIG
   ---------------------
   aws_secret
   (1 row)
   ```

2. As a superuser, create a USER storage location on the same bucket that holds the data.

   ```sql
   => CREATE LOCATION 's3://datalake' SHARED USAGE 'USER' LABEL 's3user';
   ```

   The only storage locations you can create on S3 are USER locations. You must create it as a shared location.

3. As a superuser, give access to the roles or users who will run queries against external tables.

   ```sql
   --- If you have not already created a role:
   => CREATE ROLE ExtUsers;
   --- Assign users to this role using GRANT (Role).
   => GRANT READ ON LOCATION 's3://datalake' TO ExtUsers;
   ```

4. As a user who has that role, create the external table in the usual way.
CREATE EXTERNAL TABLE sales (itemID INT, date DATE, price FLOAT)
AS COPY FROM 's3://datalake/sales/*.parquet' PARQUET;

See Creating External Tables for more information about creating and using external tables. If your data is in ORC or Parquet format, you can take advantage of partitioning to limit the data that Vertica reads during query execution. See Using Partition Columns.
Loading Data from an S3 Bucket

You can load data from an S3 bucket using COPY, for data in Parquet, ORC, text, or delimited formats. For other formats, you must use the S3 UDSourc.

If your data is in a supported format, use COPY to load it. The performance of COPY is much better than that of the UDSourc.

Important: When loading data from S3, the maximum key length is 1K (1024 bytes). The input url argument for the S3 source (which can contain multiple delimited URLs) can be no larger than 1MB.

Using COPY

Before you can read data from S3, you must create an IAM role for your EC2 instances to use, and grant that role permission to access your S3 resources. For more information about IAM roles, see Amazon's Web Services documentation.

You must also set the AWSRegion configuration parameter to your S3 bucket's region. If you do not set the correct region, you might experience a delay before the load fails because Vertica retries several times before giving up.

To load data from buckets in different regions, change the value between loads.

The following example shows how to load data from buckets in different regions.

```sql
=> SELECT SET_CONFIG_PARAMETER('AWSRegion','us-west-1');
=> COPY sales FROM 's3://AWS_DataLake/sales.parquet' PARQUET;
=> SELECT SET_CONFIG_PARAMETER('AWSRegion','us-east-1');
=> COPY sales FROM 's3://AWS_Data_1/sales.orc', 's3://AWS_Data_2/sales.orc' ORC;
```

For more information, see Specifying COPY FROM Options.

Using the S3 UDSourc

If your data is in a format that COPY from S3 does not support, you can load it using the Vertica AWS Library.
The following example illustrates the steps to define AWS credentials in order to use the APIs, and shows an example of using the S3 function in a COPY command.

1. Define the AWS credentials associated with the S3 bucket.

```sql
=> SELECT AWS_SET_CONFIG('aws_id', 'EXAMPLE');
AWS_SET_CONFIG
------------------------
aws_id
(1 row)
=> SELECT AWS_SET_CONFIG('aws_secret', 'EXAMPLE/EXAMPLEKEY');
AWS_SET_CONFIG
-------------------------
aws_secret
(1 row)
```

Alternatively, define AWS credentials using ALTER SESSION:

```sql
=> ALTER SESSION SET UDPARAMETER FOR awslib aws_id='EXAMPLE';
ALTER SESSION
=> ALTER SESSION SET UDPARAMETER FOR awslib aws_secret='EXAMPLE/EXAMPLEKEY';
ALTER SESSION
```

2. If the S3 bucket is not in the default region, define the region in the same way:

```sql
=> SELECT AWS_SET_CONFIG('aws_region', 'us-west-2');
```

Alternatively, use ALTER SESSION as shown in the previous step.

3. Create a table, or use a table already in the database.

```sql
=> CREATE TABLE NYS_Tickets (Violation_Charged_Code varchar(10), Violation_Description varchar(150), Year INT, Month INT, DOW varchar(10), Age_at_Violation INT, Gender varchar(1), State_of_Lic varchar(30), Police_Agency varchar(100), Court varchar(150), Source varchar(20));
CREATE TABLE
```

4. Use the S3 function with the SOURCE parameter in a COPY statement to load a single data file.

```sql
=> COPY NYS_Tickets WITH SOURCE S3(url='S3://examplebucket/NYS_tix/NYS_Tickets.csv') DELIMITER ',';
Rows Loaded
```
You can also load many files using a glob. This is useful when you have many files in the same S3 bucket.

```sql
=> COPY NYS_Tickets WITH SOURCE S3(bucket='S3://examplebucket/NYS_tix/*') DELIMITER ',';
Rows Loaded
----------
 14224866
(1 row)
```

The Vertica AWS Library require that you set your AWS credentials in each new session.

See Bulk Loading and Exporting Data From Amazon S3 in the Administrator's Guide for more information about using the S3 UDSource.

## Verifying the Load

You can verify that ROS files were created in the communal storage location using the AWS CLI:

```bash
$ aws s3 ls S3://testdb/
  PRE 024/
  PRE 028/
  PRE 030/
  PRE 034/
  PRE 036/
.
.
.
```

You can also use the AWS web interface by navigating to the bucket and viewing its contents.
Manage an Eon Mode Beta Database in MC

Vertica Management Console with Provisioning is a deployment of Management Console (MC), a database health and activity monitoring tool. MC provides in-browser wizards you can follow to deploy Vertica cluster instances and create an Eon Mode Beta database on them.

You can use MC to manage and monitor your database and its resources. See Using Management Console for the full guide on how to use MC to monitor your database.

When you create an Eon Mode Beta database using Vertica Management Console with Provisioning, MC detects that your database is running in Eon Mode Beta. When it does so, MC provides you with some tasks and monitoring tools that are specific to Eon Mode Beta.

Access MC

The URL for MC includes the IP address of the host on which MC is installed, followed by the MC port:

https://xx.xxx.xxx.xxx:5450/webui/

If you created your database using the using Vertica Management Console with Provisioning, MC is already fully deployed. To locate the link through the AWS Console, go to the CloudFormation Stacks page. The URL is listed under your stack's Outputs tab, labeled Management Console.

Log into MC using the MC credentials you created during installation. After login, MC displays the MC Home page.

To get started in MC with a few important database tasks, see Fast Tasks.

Manage Your Cluster

Use the MC to manage clusters that are on an AWS environment, without having to go through the AWS console or a command line.

To see all monitored clusters on the Infrastructure page, on the MC Home page click View Your Infrastructure. To view the Cluster page, click the cluster you want to manage and click Manage in the dialog that appears.
On Cluster page, you can:

- Stop, start, reboot, add, or terminate individual instances in your cluster.
- Stop, start, or reboot the entire cluster.
- Terminate the database. This terminates all cluster instances, and all AWS resources associated with the cluster.

For more detail on these cluster management tasks, see Viewing and Managing Your Cluster.

## Monitor Storage Usage and Subscription Status

To view storage usage and subscription charts for any Eon Mode Beta database you monitor, click View Your Infrastructure on the MC Home page. Then click the Storage View tab.

Use the Storage View tab in MC to monitor the size and storage capacity of your databases. You can use this page to determine if you should downsize databases that are taking up more of your storage capacity and costs, or if you need to increase storage resources for a database that has been growing.

The Storage View tab also gives insight into Eon Mode Beta database node-shard subscriptions. Subscriptions verify that your database is K-safe, and control your nodes' readiness to serve queries.

To see bar chart visualizations for subscriptions, click the Details link for any Eon Mode Beta database listed on the Storage View tab.

To learn more about monitoring storage in MC, see Monitoring Database Storage. To understand how to read the subscription charts in MC, go to Monitoring Subscription Status in Eon Mode Beta.

## Monitor Depot Activity

The depot improves query performance by preventing your queries from having to go to communal storage every time you run a query. Instead, the queries access the depot, which contains local copies of your data.
Use the Depot Activity page to monitor query activity, churn, and settings for your database's depot. To view this page, click the Activity tab at the bottom of your database's Overview page, then select Depot Activity Monitoring from the chart list.

See Monitoring Depot Activity in MC for more on monitoring using this page.

**See Also**

- Using Management Console
- Fast Tasks
Shutting Down and Reviving a Database

You can stop a database running in Eon Mode Beta and then, later, revive it into a newly provisioned cluster and continue where you left off. You do not need to wait for nodes to recover or data to be moved; your cluster is ready to use as soon as you revive it.

Vertica maintains an up-to-date version of your Eon Mode Beta database's metadata and data in its communal storage location. Communal storage resides in a persistent AWS S3 location, so even if you terminate cluster hosts, the database can later be revived from its communal storage location, including its metadata.

When you stop a cluster whose hosts use instance storage where data is not persistently stored, stopping the entire cluster removes the database from the hosts. Terminating the cluster also removes the database. If it was an Eon Mode Beta database, you can revive it on a restarted or newly provisioned cluster using admintools.

For example, if your Eon Mode Beta database will not be queried during a long period of time, you might terminate the cluster to save computing costs. To bring the database back into use, provision a new cluster and revive your Eon Mode Beta database on it.

Synchronize Metadata

When maintaining Vertica in Eon Mode Beta, Vertica stores an up-to-date version of the catalog, which contains the database's metadata, in communal storage.

When an Eon Mode Beta database is running, Vertica automatically syncs the catalog every five minutes. Therefore it is not typically necessary to intervene in the synchronization process. However, before shutting down it is a good idea to make sure that the persistent copy of the catalog contains all recent changes. You can use the following processes to make sure the database's catalog will be up to date when you revive.

Use the TxnLogSyncInterval parameter to set a time interval for which the metadata is pushed to communal storage (the default is 5 minutes):

```sql
=> ALTER DATABASE mydb SET TxnLogSyncInterval = 10 minutes
```

You can also force an immediate push to communal storage at any time using the following command:

```sql
=> select hurry_service('SYSTEM','TxnLogSyncTask');
```
You can use the sync_catalog function to immediately synchronize the catalog on all nodes or a specific node.

```sql
=> select sync_catalog();
```

## Revive the Database using Admintools

Use the following process to revive your Eon Mode Beta database on an existing cluster using admintools.

**Caution:** Using the same communal storage location for multiple running databases causes data corruption. To avoid corruption, never use the revive functionality to simultaneously run the same Eon Mode Beta database on different clusters.

**Prerequisites:**

- Communal storage location (an AWS S3 bucket) of the *stopped* Eon Mode Beta database you plan to revive
- Username and password of the Eon Mode Beta database you plan to revive
- AWS cluster with no other running databases, whose hosts are in the same region as your communal storage location's S3 bucket (To use a cluster in a new region, you must first clone the previous communal storage location to an S3 bucket in the new region)
- Amazon key pair for SSH access to an instance in the AWS cluster
- IP addresses of all hosts on the cluster

### Revive the Database

To revive the database, run the following:

```bash
$ admintools -t revive_db --communal-storage-location=s3://<directory> -s $HOSTLIST --force -d $DATABASENAME
```
Welcome to the Vertica on the Cloud guide. This section explains how you can create Vertica clusters on different cloud platforms.

This document assumes that you are familiar with the cloud environment on which you will create your Vertica cluster.
Vertica on Amazon Web Services

Welcome to the Vertica on Amazon Web Services (AWS) guide. This section explains how you can create and manage Vertica clusters on AWS.

In This Section

Overview of Vertica on Amazon Web Services (AWS)

Vertica clusters on AWS operate on Amazon Machine Instances (AMI). The instructions in this document apply to AMIs built with Vertica Version 9.0.x.

Key Topics

The following are links to key topics within the Vertica on AWS documentation:

- If you created a standard 3 node cluster using a cloud formation template and are looking for information about how to connect, visit Connecting to an Instance.

- If you have a running cluster, and want to add nodes to it, visit Adding Nodes to a Running AWS Cluster.

- If you are looking for information about backup and restore operations, visit Backup and Restore Vertica on AWS.

Packages

Vertica AMIs come with the following Vertica packages pre-installed:

- Geospatial Analytics

- Vertica Pulse
Supported Instance Types

Vertica supports a range of Amazon Web Services (AWS) instance types, each optimized for different purposes. Choose the instance type that best matches your performance and price needs as a user:

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose</td>
<td>m4.4xlarge</td>
</tr>
<tr>
<td></td>
<td>m4.10xlarge</td>
</tr>
<tr>
<td>Compute</td>
<td>c3.4xlarge</td>
</tr>
<tr>
<td></td>
<td>c3.8xlarge</td>
</tr>
<tr>
<td></td>
<td>c4.4xlarge</td>
</tr>
<tr>
<td></td>
<td>c4.8xlarge</td>
</tr>
<tr>
<td>Memory</td>
<td>r3.4xlarge</td>
</tr>
<tr>
<td></td>
<td>r3.8xlarge</td>
</tr>
<tr>
<td></td>
<td>r4.4xlarge</td>
</tr>
<tr>
<td></td>
<td>r4.8xlarge</td>
</tr>
<tr>
<td></td>
<td>r4.16xlarge</td>
</tr>
<tr>
<td>Storage (supports ephemeral storage)</td>
<td>i2.4xlarge</td>
</tr>
<tr>
<td></td>
<td>i2.8xlarge</td>
</tr>
<tr>
<td>Dense-storage (supports ephemeral storage)</td>
<td>d2.4xlarge</td>
</tr>
<tr>
<td></td>
<td>d2.8xlarge</td>
</tr>
</tbody>
</table>

Note: Data stored on the ephemeral drives of a storage-optimized instance exists only while that instance is powered on. After powering off a storage-optimized system, data on ephemeral drives is lost.

More Information

For more information about Amazon cluster instances and their limitations, see the Amazon documentation.
Vertica AMI Operating Systems

Vertica provides Vertica and Management Console (MC) AMIs in multiple operating systems. Vertica AMIs are available in the following operating systems:

- Red Hat 7.3
- Centos 7.3

MC AMIs are available in the following operating systems:

- Red Hat 7.3
- Centos 7.3

Vertica AMI Sleep C-States

By default, the following instances have their processor C-states set to a value of 1 in the Vertica AMI:

- c4.8xlarge
- d2.8xlarge
- m4.10xlarge

This measure is meant to improve performance by limiting the sleep states an instance running Vertica will use.

More Information

For more information about sleep states, visit the AWS Documentation.

EC2 Launch Options

You can create one or multi node Vertica clusters on EC2 instances. Search for Vertica in the Amazon Marketplace to view these launch options.
1-Click Launch
This option deploys a single-node Vertica AMI with default settings.

Manual Launch
This option deploys a single-node Vertica AMI or a 3-node Vertica cluster with more customizable options.

Service Catalog
This option deploys a single-node AMI or a multi-node cluster for an administrator to create and distribute to end users.

**AWS Features That Work with Vertica**

Vertica works with a number of AWS features.

Enhanced Networking
Vertica recommends that you use the AWS enhanced networking for optimal performance.
For details, see [Enabling Enhanced Networking on Linux Instances in a VPC](https://aws.amazon.com/documentation/vertica/) in the AWS documentation.

Command Line Interface
You can use the the Amazon command-line Interface (CLI) with your Vertica AMIs.
For information about the Amazon CLI, visit the [AWS Documentation](https://aws.amazon.com/documentation/vertica/).
Elastic Load Balancing

On your Vertica AMI, you can use elastic load balancing (ELB) for queries up to one hour. When enabling ELB, ensure that the timer is configured to 3600 seconds.

For information about ELB, refer to Amazon documentation.

Configure Your Network

Before you create your cluster, you must configure the network on which Vertica will run. Vertica requires a number of specific network configurations to operate on AWS. You may also have specific network configuration needs beyond the default Vertica settings.

The following sections explain which Amazon EC2 features you need to configure for instance creation.

Creating a Placement Group

Create a placement group to ensure that your nodes will be properly co-located.

More Information

For information about what a placement group is, as well as how to create one, visit the AWS documentation.

Creating a Key Pair

Use a key pair to perform the following:

- Authenticate your connection as dbadmin to your instances from outside your cluster.
- Install and configure Vertica on your AWS instances.

The AWS documentation explains what a key pair is and how to create one.
Creating a Virtual Private Cloud

A Vertica cluster on AWS must be logically located in the same network. This is similar to placing the nodes of an on-premises cluster within the same network. Create a virtual private cloud (VPC) to ensure the nodes in your cluster will be able to communicate with each other within AWS.

Create a single public subnet VPC with the following configurations:

- Assign a Network Access Control List (ACL) that is appropriate to your situation. The default ACL does not provide a high level of security.

- Enable DNS resolution and enable DNS hostname support for instances launched in this VPC.

- Add the required network inbound and outbound rules to the Network ACL associated to the VPC.

Note: A Vertica cluster must be operated within a single availability zone.

More Information

For information about a VPC, including how to create one, visit the AWS documentation.

Network ACL Settings

Vertica requires the following basic network access control list (ACL) settings on an AWS instance running the Vertica AMI. Vertica recommends that you secure your network with additional ACL settings that are appropriate to your situation; the default ACL does not provide a high level of security.

Inbound Rules

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Use</th>
<th>Source</th>
<th>Allow/Deny</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>TCP (6)</td>
<td>22</td>
<td>SSH (Optional -- for access to your cluster from outside your VPC)</td>
<td>User Specific</td>
<td>Allow</td>
</tr>
<tr>
<td>Type</td>
<td>Protocol</td>
<td>Port Range</td>
<td>Use</td>
<td>Source</td>
<td>Allow/Deny</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>Custom TCP Rule</td>
<td>TCP (6)</td>
<td>5450</td>
<td>MC (Optional -- for MC running outside of your VPC)</td>
<td>User Specific</td>
<td>Allow</td>
</tr>
<tr>
<td>Custom TCP Rule</td>
<td>TCP (6)</td>
<td>5433</td>
<td>SQL Clients (Optional -- for access to your cluster from SQL clients)</td>
<td>User Specific</td>
<td>Allow</td>
</tr>
<tr>
<td>Custom TCP Rule</td>
<td>TCP (6)</td>
<td>50000</td>
<td>Rsync (Optional -- for backup outside of your VPC)</td>
<td>User Specific</td>
<td>Allow</td>
</tr>
<tr>
<td>Custom TCP Rule</td>
<td>TCP (6)</td>
<td>1024-65535</td>
<td>Ephemeral Ports (Needed if you use any of the above)</td>
<td>User Specific</td>
<td>Allow</td>
</tr>
<tr>
<td>ALL Traffic</td>
<td>ALL</td>
<td>ALL</td>
<td>N/A</td>
<td>0.0.0.0/0</td>
<td>Deny</td>
</tr>
</tbody>
</table>

**Outbound Rules**

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Use</th>
<th>Source</th>
<th>Allow/Deny</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom TCP Rule</td>
<td>TCP (6)</td>
<td>0 - 65535</td>
<td>Ephemeral Ports</td>
<td>0.0.0.0/0</td>
<td>Allow</td>
</tr>
</tbody>
</table>

You can use the entire port range specified in the table above, or find your specific ephemeral ports by entering the following command:

```
cat /proc/sys/net/ipv4/ip_local_port_range
```

**More Information**

For detailed information on network ACLs within AWS, refer to [Amazon's documentation](https://aws.amazon.com/documentation/security/network-acl/).

For detailed information on ephemeral ports within AWS, refer to [Amazon's documentation](https://aws.amazon.com/documentation/security/ephemeral-ports/).
Creating and Assigning an Internet Gateway

When you create a VPC, an internet gateway is automatically assigned to it. You can use that gateway, or you can assign your own. If you are using the default internet gateway, continue with the next procedure, Creating a Security Group.

Otherwise, create an internet gateway specific to your needs. Associate your internet gateway with your VPC and subnet.

More Information

For information about what an internet gateway is, as well as how to create one, visit the AWS documentation.

Assigning an Elastic IP Address

An elastic IP is an unchanging IP address that you can use to connect to your cluster externally. Vertica recommends you assign a single elastic IP to a node in your cluster. You can then connect to other nodes in your cluster from your primary node using their internal IP addresses dictated by your VPC settings.

Create an elastic IP address.

More Information

For information about what an elastic IP is, as well as how to create one, visit the AWS documentation.

Creating a Security Group

The Vertica AMI has specific security group requirements. When you create a Virtual Private Cloud (VPC), AWS automatically creates a default security group and assigns it to the VPC. You can use the default security group, or you can name and assign your own.

Create and name your own security group with the following basic security group settings. You may make additional modifications based on your specific needs.
## Inbound

<table>
<thead>
<tr>
<th>Type</th>
<th>Use</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td></td>
<td>TCP</td>
<td>22</td>
<td>My IP</td>
<td>Limited IP range 169.24.165.0/24 or 0.0.0.0/0 for all internet access.</td>
</tr>
<tr>
<td>DNS (UDP)</td>
<td></td>
<td>UDP</td>
<td>53</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom UDP</td>
<td>Spread</td>
<td>UDP</td>
<td>4803 and 4804</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom TCP</td>
<td>Spread</td>
<td>TCP</td>
<td>4803</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom TCP</td>
<td>VSQL/SQL</td>
<td>TCP</td>
<td>5433</td>
<td>My IP</td>
<td>Limited IP range 169.24.165.0/24 or 0.0.0.0/0 for all internet access.</td>
</tr>
<tr>
<td>Custom TCP</td>
<td>Inter-node Communication</td>
<td>TCP</td>
<td>5434</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom TCP</td>
<td></td>
<td>TCP</td>
<td>5444</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom TCP</td>
<td>MC</td>
<td>TCP</td>
<td>5450</td>
<td>My IP</td>
<td>Limited IP range 169.24.165.0/24 or 0.0.0.0/0 for all internet access.</td>
</tr>
<tr>
<td>Custom TCP</td>
<td>Rsync</td>
<td>TCP</td>
<td>48073</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
<tr>
<td>Custom TCP</td>
<td></td>
<td>TCP</td>
<td>50000</td>
<td>My IP</td>
<td>10.0.0.0/24</td>
</tr>
</tbody>
</table>
All ports must have a rule to open in the subnetCDIR level to allow nodes to be interconnected. For example, 10.11.12.0/24.

Note: In Management Console (MC), the Java IANA discovery process uses port 7 once to detect if an IP address is reachable before the database import operation. Vertica tries port 7 first. If port 7 is blocked, Vertica switches to port 22.

### Outbound

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Destination</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>All TCP</td>
<td>TCP</td>
<td>0-65535</td>
<td>Anywhere</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>All ICMP</td>
<td>ICMP</td>
<td>0-65535</td>
<td>Anywhere</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>All UDP</td>
<td>UDP</td>
<td>0-65535</td>
<td>Anywhere</td>
<td>0.0.0.0/0</td>
</tr>
</tbody>
</table>

### More Information

For information about what a security group is, as well as how to create one, visit the [AWS documentation](https://aws.amazon.com/documentation/).  

### Installing and Running Vertica on AWS

Once you have configured your network, you are ready to create your AWS instances and install vertica. Follow these procedures to install and run Vertica on AWS.

### Configuring and Launching an Instance

Once you have configured your network settings on AWS, you are ready to configure and launch the instances that you will later install Vertica onto. An Elastic Compute Cloud (EC2) instance without a Vertica AMI is similar to a traditional host. Just like with an on-premises
cluster, you must prepare and configure your cluster and network at the hardware level before you can install Vertica.

When you create an EC2 instance on AWS using a Vertica AMI, the instance includes the Vertica software and the recommended configuration. The Vertica AMI acts as a template, requiring fewer configuration steps. Vertica recommends that you use the Vertica AMI as is—without modification.

Using the network configurations you created earlier, configure an instance in AWS:

1. Select the Vertica AMI from the AWS marketplace.
2. Select a supported instance type.
3. Specify the number of instances you want to launch. A Vertica cluster uses identically configured instances of the same type. You cannot mix instance types.
4. Choose your VPC.
   
   Note: Not all data centers support VPC. If you receive an error message that states "VPC is not currently supported...", choose a different region and zone (for example, choose us-east-1e rather than us-east-1c).
5. Assign a Placement Group.

Add Storage:

Add storage to your instances based on your needs. Consider the following:

- Add a number of drives equal to the number of physical cores in your instance. For example, for a c3.8xlarge, add eight drives. For an r3.4xlarge, add four drives.

- Do not store your information on the root volume.

- For optimal performance with EBS volumes, Amazon recommends that you configure them in a RAID 0 array on each node in your cluster.

- Create your storage when you create your instances. This allows AWS to optimize storage performance.

Configure Security Group:

1. Choose between your previously configured security group or the default security group.
2. Choose the key value pairs you intend to use with Vertica.

Launch Instances:
Verify that your instances are running.

Connecting to an Instance

Using your private key, connect to your cluster through the instance you attached an elastic IP to.

1. As the dbadmin user, type the following command, substituting your ssh key:

   ```
   # ssh --ssh-identity <ssh key> dbadmin@elasticipaddress
   ```

2. Select Instances from the Navigation panel.

3. Select the instance that is attached to the Elastic IP.

4. Click Connect.

5. On Connect to Your Instance, choose one of the following options:
   - A Java SSH Client directly from my browser—Add the path to your private key in the field Private key path, and click Launch SSH Client.
   - Connect with a standalone SSH client—Follow the steps required by your standalone SSH client.

Connecting to an Instance from Windows Using Putty

If you connect to the instance from the Windows operating system, and plan to use Putty:

1. Convert your key file using PuTTYgen.
2. Connect with Putty or WinSCP (connect via the elastic IP), using your converted key (i.e., the *ppk file).
3. Move your key file (the *pem file) to the root dir using Putty or WinSCP.

Preparing Instances

After you create your instances, you need to prepare them for cluster formation. Prepare your instances by adding your AWS .pem key and your Vertica license.
By default, each AMI includes a Community Edition license. Once Vertica is installed, you can find the license at this location:

```
/opt/vertica/config/licensing/vertica_community_edition.license.key
```

1. As the dbadmin user, copy your *pem file (from where you saved it locally) onto your primary instance.

   Depending upon the procedure you use to copy the file, the permissions on the file may change. If permissions change, the `install_vertica` script fails with a message similar to the following:

   FATAL (19): Failed Login Validation 10.0.3.158, cannot resolve or connect to host as root.

   If you receive a failure message, enter the following command to correct permissions on your *pem file:

   ```bash
   chmod 600 /<name-of-pem>.pem
   ```

2. Copy your Vertica license over to your primary instance, placing it in your home directory or other known location.

### Configuring Storage

Vertica recommends that you do not store your information on the root volume, especially your data and catalog directories. Instead, use dedicated EBS volumes for node storage. To take advantage of bursting, limit EBS volumes to 1TB or less. When configuring your storage, make sure to use a supported file system.

For best performance, you can combine multiple EBS volumes into RAID-0. Vertica provides a shell script, which automates the storage configuration process.

### Determining Volume Names

Before you combine volumes for storage, make note of your volume names so that you can alter the `configure_aws_software_raid.sh` shell script. You can find your volumes with the following commands:

```bash
cd /dev
ls
```

Your volumes start with `xvd`. 
Important: Ignore your root volume. Do not include any of your root volumes in the RAID creation process.

Combining Volumes for Storage

Follow these sample steps to combine your EBS volumes into RAID 0 using the configure_aws_software_raid.sh shell script.

1. Edit the /opt/vertica/sbin/configure_aws_software_raid.sh shell file as follows:
   a. Comment out the safety exit command at the beginning.
   b. Change the sample volume names to your own volume names, which you noted previously. Add more volumes, if necessary.

2. Run the /opt/vertica/sbin/configure_aws_software_raid.sh shell file. Running this file creates a RAID 0 volume and mounts it to /vertica/data.

3. Change the owner of the newly created volume to dbadmin with chown.

4. Repeat steps 1-3 for each node on your cluster.

More Information

For more information about EBS storage, refer to the Amazon documentation.

Forming a Cluster

Use the install_vertica script to combine two or more individual instances and create a cluster.

Check the My Instances page for a list of current instances and their associated IP addresses. You need these IP addresses when you run the install_vertica script.

Combining Instances

The following example combines instances using the install_vertica script.
While connected to your primary instance, enter the following command to combine your instances into a cluster. Substitute the IP addresses for your instances and include your root *pem file name.

```
sudo /opt/vertica/sbin/install_vertica --hosts 10.0.11.164,10.0.11.165,10.0.11.166 --dba-user-password-disabled --point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-pem>.pem --license <license.file>
```

**Note:** If you are using Community Edition, which limits you to three instances, you can simply specify -L CE with no license file.

When you issue `install_vertica` or `update_vertica` on a Vertica AMI script, --point-to-point is the default. This parameter configures Spread to use direct point-to-point communication between all Vertica nodes, which is a requirement for clusters on AWS.

Once you have combined your instances, Vertica recommends deleting your *pem key from your cluster with the shred command to reduce security risks:

```
shred examplekey.pem
```

**Important:** You will need your .pem key to perform future Vertica updates.

Once you have formed a cluster, [Create a database](#).

### Considerations When Using the install_vertica or update_vertica Scripts

- By default, the installer assumes that you have mounted your storage to `/vertica/data`. To specify another location, use the --data-dir argument. Vertica does not recommend that you store your data on the root drive.

- Password logons present a security risk on AWS. Include the parameter --dba-user-password-disabled so that the installer does not prompt for a password for the database user.

For complete information on the `install_vertica` script and its parameters, see the Installation Guide, specifically the section, *About the install_vertica Script*.

### After Your Cluster Is Up and Running

Once your cluster is configured and running:
1. **Create a database.** When Vertica was installed for you, a Vertica database administrator was created, dbadmin. You can use this pre-created dbadmin user to create and start a database. Refer to the Vertica Installation Guide for information on the dbadmin administrator.

2. **Configure a database.** Refer to the Vertica Administrator’s Guide for information on configuring a database.

3. **Refer to the full documentation set for Vertica for other tasks.**

### Safe Shutdown

To safely stop or reboot your cluster:

1. Stop the database.

2. Stop or reboot one or more instances.

**Caution:** If you stop or reboot an instance (or the cluster) without shutting the database down first, disk or database corruption could result. Shutting the database down first ensures that Vertica is not in the process of writing to disk when you shutdown. Refer to the Vertica Administrator’s Guide for information on stopping a database.

### Using Management Console (MC) on AWS

Management Console (MC) is a database management tool that allows you to view and manage aspects of your cluster. As of Vertica version 8.0.1, Vertica provides an MC AMI, which you can use with AWS.

The MC AMI allows you to create an instance, dedicated to running MC, that you can attach to a new or existing Vertica cluster on AWS.

You can create and attach an MC instance to your Vertica on AWS cluster at any time, if your AWS cluster is running Vertica version 8.0.1 or later.

### See Also

- [Network ACL Settings](#)
Configuring and Launching an MC Instance

When you create an EC2 instance on AWS using a Vertica MC AMI, the instance includes a complete installation of Vertica MC. Micro Focus recommends that you use the Vertica MC AMI as is—without modification.

Configure and Launch an MC Instance in AWS:

1. Select the Vertica MC AMI from the AWS marketplace.
2. Create a single instance, Vertica recommends c3.large for most implementations.
3. Place your MC instance within the VPC, subnet, and security group of the cluster which it will monitor.
4. Choose the key value pairs you use with Vertica.
5. Assign an elastic IP to the instance to make the MC externally accessible.

Configure Your Security Group:

The MC AMI requires specific security group configurations to communicate with the Vertica cluster. Enable the security rules for the MC AMI listed under the MC section of the Network ACL Settings section.

Launch MC Instance:

Verify that your MC instance is running.

Logging in to MC and Managing Your Cluster

After you launch your MC instance and configured your security group settings, log in to your database. To do so, use the elastic IP you specified during instance creation.

From this elastic IP, you can manage your Vertica database on AWS using standard MC procedures.
Considerations When Using MC on AWS

- Because MC is already installed on the MC AMI, the MC installation process does not apply.
- To uninstall MC on AWS, follow the procedures provided in Uninstalling Management Console before terminating the MC Instance.

Related Topics

- Refer to Using Management Console for information.
- Refer to Managing Database Clusters to create or import your cluster.

Adding Nodes to a Running AWS Cluster

Use these procedures to add instances/nodes to an AWS cluster. The procedures assume you have an AWS cluster up and running and have most-likely accomplished each of the following:

- Created a database.
- Defined a database schema.
- Loaded data.
- Run the database designer.
- Connected to your database.

Related Topics

- Launching New Instances to Add to an Existing Cluster
- Including New Instances as Cluster Nodes
- Adding Nodes and Rebalancing the Database
Launching New Instances to Add to an Existing Cluster

Perform the procedure in Configuring and Launching an Instance to create new instances that you then will add to your existing cluster. Be sure to choose the same details you chose when you created the original instances (VPC, placement group, subnet, and security group).

Including New Instances as Cluster Nodes

The Instances page lists the instances and their associated IP addresses. You need the IP addresses when you run the install_vertica script.

If you are configuring EBS volumes, be sure to configure the volumes on the node before you add the node to your cluster.

To add the new instances as nodes to your existing cluster:

1. Configure and launch your new instances.
2. Connect to the instance that is assigned to the Elastic IP. See Connecting to an Instance if you need more information.
3. Run the vertica installation script to add the new instances as nodes to your cluster. Specify the internal IP addresses for your instances and your *pem file name.

   ```
   sudo /opt/vertica/sbin/install_vertica --add-hosts <instance-ip> --db-user-password-disabled --point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-pem>.pem
   ```

Adding Nodes and Rebalancing the Database

Once you have added the new instances to your existing cluster, you add them as nodes to your cluster, and then rebalance the database.

Follow the procedure given in the Administration Guide, Adding Nodes to a Database.

Removing Nodes From a Running AWS Cluster

Use these procedures to remove instances/nodes from an AWS cluster.
Related Topics

- Preparing to Remove a Node
- Removing Hosts from the Database
- Removing Nodes from the Cluster
- Stopping the AWS instance

Preparing to Remove a Node

Back up the Database. See the Backup and Restore sections of this document for more information.

Vertica recommends that you back up the database before performing this significant operation because it entails creating new projections, deleting old projections, and reloading data.

Removing Hosts From the Database

Before performing the procedure in this section, you must have completed the tasks referenced in Preparing to Remove a Node. The following procedure assumes that you have both backed up your database and lowered the k-safety.

Note: Do not stop the database.

Perform the following to remove a host from the database.

1. While logged on as dbadmin, launch Administration Tools.

   $ /opt/vertica/bin/admintools

   Note: Do not remove the host that is attached to your EIP.

2. From the Main Menu, select Advanced Tools Menu.

3. From Advanced Menu, select Cluster Management. Select OK.

4. From Cluster Management, select Remove Host(s). Select OK.
5. From Select Database, choose the database from which you plan to remove hosts. Select OK.

6. Select the host(s) to remove. Select OK.

7. Click Yes to confirm removal of the hosts.

   **Note:** Enter a password if necessary. Leave blank if there is no password.

8. Select OK. The system displays a message letting you know that the hosts have been removed. Automatic re-balancing also occurs.

9. Select OK to confirm. Administration Tools brings you back to the Cluster Management menu.

Removing Nodes From the Cluster

To remove nodes from a cluster:

run the `install_vertica` script and specify:

- The option `--remove-hosts` followed by the IP addresses of the nodes you are removing
- The option `--ssh-identity` followed by the location and name of your *pem* file. (The following example removes only one node from the cluster.)
- The option `--dba-user-password-disabled`

The following example removes 1 node from the cluster:

```
sudo /opt/vertica/sbin/install_vertica --remove-hosts 10.0.11.165 --point-to-point --ssh-identity ~/<name-of-pem>.pem --dba-user-password-disabled
```

Stopping the AWS Instances (Optional)

Once you have removed one or more nodes from your cluster, to save costs associated with running instances, you can choose to stop or terminate the AWS instances that were previously part of your cluster. This step is optional because, once you have removed the node from your Vertica cluster, Vertica no longer sees the instance/node as part of the cluster even though it is still running within AWS.

To stop an instance in AWS:
1. On AWS, navigate to your Instances page.
2. Right-click on the instance, and choose Stop.

Upgrading and Migrating Vertica on AWS

The topics in this section explain how you upgrade your version of Vertica and migrate data between clusters on AWS.

Upgrading to the latest Version of Vertica on AWS

Use these procedures for upgrading to the latest version of Vertica on AWS. The procedures assume that you have a 7.1 or later cluster successfully configured and running on AWS. If you are setting up a Vertica cluster on AWS for the first time, follow the detailed procedure for installing and running Vertica on AWS.

Note: Both install_vertica and update_vertica use the same parameters.

Related Topics

- Preparing to Upgrade Your Cluster
- Upgrading Vertica on AWS

Preparing to Upgrade Your Cluster

Prepare for the upgrade to the latest Vertica version by performing the following:

1. Back up your existing database. See the Backup and Restore section for more information.
2. Download the Vertica install package. See Download and Install the Vertica Install Package in the Installation Guide for more information.
Upgrading Vertica Running on AWS

Vertica supports upgrades of Vertica Server running on AWS instances created from the Vertica AMI. To upgrade Vertica, follow the instructions provided in the Vertica upgrade documentation.

Add the following arguments to the upgrade script:

- --dba-user-password-disabled
- --point-to-point

Migrating to Vertica 7.0 or later on AWS

Note: If you had a Vertica installation running on AWS prior to Release 6.1.x, you can migrate to Vertica 9.0.x or later using a new preconfigured AMI.

For more information, see the Solutions tab of the myVertica portal.

Migrating Data Between AWS Clusters

This section provides guidance for copying (importing) data from another AWS cluster, or exporting data between AWS clusters.

There are three common issues that occur when exporting or copying on AWS clusters. The issues are listed below. Except for these specific issues as they relate to AWS, copying and exporting data works as documented in the Administrator’s Guide section, Copying and Exporting Data.

1. Ensure that all nodes in source and destination clusters have their own elastic IPs (or public IPs) assigned.
   If your destination cluster is located within the same VPC as your source cluster, proceed to step 3. Each node in one cluster must be able to communicate with each node in the other cluster. Thus, each source and destination node needs an elastic IP (or public IP) assigned.

2. Set the parameter DontCheckNetworkAddress to true.
   On AWS, when creating a network interface, you receive an error if you attempt to assign
the elastic IP to an AWS node (example uses a sample elastic IP address):
dbadmin=> CREATE NETWORK INTERFACE eipinterface ON v_tpch_node0001
     with '107.23.151.10';

ERROR 4125: No valid address found for [107.23.151.10] on this node
This error occurs because the elastic IP is the public IP and not the private IP of the target node. To resolve this issue, first set the parameter DontCheckNetworkAddress to true:
select set_config_parameter('DontCheckNetworkAddress','1');
You can find information on the CREATE NETWORK INTERFACE statement and SET_CONFIG_PARAMETER in the SQL Reference Manual.

3. Ensure your security group allows the AWS clusters to communicate.
   Check your security groups for both your source and destination AWS clusters. Ensure that ports 5433 and 5434 are open. If one of your AWS clusters is on a separate VPC, ensure that your network access control list (ACL) allows communication on port 5434.
   
   Note: This communication method exports and copies (imports) data through the internet. You can alternatively use non-public IPs and gateways, or VPN to connect the source and destination clusters.

**Backup, Restore, and Replication on Amazon Web Services**

The Vertica backup and restore utility, vbr, is fully compatible with AWS. All vbr commands function properly on AWS.

**More Information**

For more information about backups, visit the Backing Up and Restoring the Database section of the Vertica documentation.

For information on replication, refer to Copying the Database to Another Cluster and Replicating Tables and Schemas to an Alternate Database.

For more information about backup host configuration, including passwordless SSH configuration, refer to Configuring Backup Hosts.
K-Safe Cluster Configuration

You can protect against isolated node failures with a K-safe cluster configuration. A K-safe cluster stores buddy data on other nodes in the cluster to prevent data loss if a node fails. For a cluster to be K-safe, it must consist of at least 3 nodes.

To learn more about K-safe cluster configurations, visit the Designing For K-Safety section of the Vertica documentation.

Recover from a K-Safe Failure

To recover from an isolated node failure, your cluster must still be in the UP state. Recovering from a K-safe failure does not require a backup.

If a node goes down and you cannot re-connect to it, you must re-create the node. Follow these steps to re-create a source node and recover from a K-safe failure:

- Create a Target Node
- Recover a Node

Note: If your primary node fails, you must reassign your Elastic IP, and then copy your .pem key file to one of your other running nodes to recover.

Create a Target Node

Create a target AWS instance ensuring the following:

- Subnet, network, and VPC—Your target instance must be in the same subnet, network, and VPC. It must also have the same network configurations as the nodes in your source cluster.

- Cluster placement and availability zone—Your target instance must be in the same cluster placement group and availability zone as the nodes in your source cluster.

- IP address—The internal IP address of your target instance must be the same as the IP address of the source node. Use the Network Interfaces option during instance creation to assign internal IP addresses.
Version compatibility—Your target instance must use the same Vertica AMI version and hotfix version as the nodes in your source cluster.

Instance type—Your target instance must use the same instance type as the nodes in your source cluster.

Recover a Node

1. On all your nodes, delete the .pem key information for the failed source node in the following locations:
   - /root/.ssh/known_hosts
   - /home/dbadmin/.ssh/known_hosts

2. On your main node, run the install script install_vertica, specifying:
   - your own key file
   - -Y option as point-to-point
   - dba user password disabled
   - dba user dbadmin:
     ```
sudo /opt/vertica/sbin/install_vertica -i ~/userkey.pem -Y --point-to-point --dba-user-password-disabled --dba-user dbadmin
     ```

3. Connect to your target node with SSH, and configure its storage to match your source cluster's storage.

4. Configure your target node:
   a. Create an empty catalog and data directory matching the source node:
      ```
      mkdir -p /vertica/data/VMart/v_vmart_node0003_catalog
      ```
   b. Change the owner of the catalog and data directory to verticadba:
      ```
      sudo chown dbadmin:verticadba /vertica/data/VMart
      ```

5. Restart the target node using admintools, specifying the IP address of your target node and your database name:
   ```
   admintools -t restart_node -s 10.0.10.15 -d VMart
   ```

Your target node will now recover using data from its buddy, making your cluster K-safe once again. Depending on the size of your database, recovery may take some time.
Full Backup

A full backup captures a complete image of your database at a specific point in time. This option is the safest and most stable backup approach. A full backup lets you recover from a non K-safe loss, such as a multi-node failure or a total cluster failure.

**Backup Artifacts**
1. EBS snapshots (one for each node)
2. /opt/vertica/conf/admintools.conf
3. vmart_backup.ini
Storage Considerations

When you choose storage for your full backup, be aware of these considerations:

- If you are using ephemeral storage, you must perform a full backup.
- You must use EBS storage for the backup volume.
- To reduce costs, you may choose cheaper, slower EBS storage drive types for the backup volume.

To perform a full backup, you need the following:

- Properly formatted and mounted backup volumes
- An AWS snapshot of your backup volumes
- A backup configuration file

Find more information about full backups in the Types of Backups section of the Vertica documentation.

Related Topics

The process of creating a full backup on AWS requires the following tasks:

- Prepare Backup Volumes
- Perform a Full Backup

Prepare Backup Volumes

1. Find the size of the data catalog on each source node:
   
   ```bash
df -h /vertica/data/
   ```
   
   Look for the number in the Used column.

2. Find the largest data catalog of all the nodes in your source cluster, and add a 20–50% safety margin.
3. Create and mount a new EBS volume to each source node. This volume should equal your largest data catalog with its safety margin.

4. Verify that your newly mounted volumes appear on each source node:
   ```
   ls /dev
   ```

5. Format all volumes, using a Vertica supported file system:
   ```
   sudo mkfs.ext4 /dev/xvdf
   ```

6. Create the backup folder. Specify the location where you mount the new backup volume:
   ```
   sudo mkdir /vertica/backup
   ```

7. Mount the volume to /vertica/backup, and make it persistent:
   ```
   sudo mount /vertica/backup
   sudo bash -c "echo '/dev/xvdf /vertica/backup ext4 defaults 0 0' > /etc/fstab"
   ```

8. Set dbadmin as the owner for all /vertica/backup:
   ```
   sudo chown dbadmin:verticadba /vertica/backup
   ```

9. Verify the success of the mount operation by entering:
   ```
   df -h
   ```

10. Repeat the formatting and mounting steps on all nodes.

11. Create a Backup configuration file. Vertica includes several sample files that you can use as a starting point.

12. Once all of your backup locations exist, initialize those locations using the vbr init task:
    ```
    vbr -t init -c config_file_name
    ```

Once your volumes are prepared, perform a backup as you would normally.

**Perform a Full Backup**

The process for performing a full backup on AWS is the same as it is for any supported file system.

Your backup configuration file, and your initial backup contain a complete backup of your database. Subsequent backups contain only data that has changed from the preceding backup. You can perform a complete restore from any backup.
Note:

- Incremental snapshots can take different amounts of time compared to the initial snapshot, because only the differences from the first snapshot are saved. If the differences are few, subsequent backups and snapshots can be quite fast.
- Snapshots are taken asynchronously. Therefore, you can continue to write the next backup as the previous snapshot completes.

For more information on EBS snapshots, visit the AWS documentation.

Restore from a Full Backup

If a cluster fails, use your backup configuration file and backup volume snapshots to restore your cluster from your last full backup.

Related Topics

The process of restoring from a full backup on AWS requires the following tasks:

- Create a Target Cluster
- Restore a Database from a Full Backup

Create a Target Cluster

Create a target cluster on AWS. Your new instances must match your source cluster in the following ways:

- Network and VPC—The target instances must be in the same network and VPC as each other. However, they do not have to be in the same network and VPC as your source cluster.
- Number of instances and nodes—You must use the same number of instances and nodes as your source cluster.
- IP addresses—The internal IP addresses of the target nodes must be the same as the internal IP addresses of the nodes in the source cluster.
Version compatibility—Your target cluster must be running the same AMI and Vertica versions and using the same hotfix version as your source cluster.

Your new instances may differ from the source cluster in the following ways:

- Your target cluster may be in a different availability zone.
- You may use a different instance type for your target cluster.

Restore a Database from a Full Backup

After you create a target cluster, you must restore the database that was on the source cluster:

1. Using your source cluster backup snapshots, create and attach one volume for each node in your cluster and attach them to the nodes in your target cluster. Verify that the snapshots from the source cluster match with their respective nodes on the target cluster.

   Important: You must use the correct device mapping. For example, a backup snapshot taken for node 1, must be recreated in the new cluster at node 1.

2. Mount the backup location on all nodes of the target cluster with the same file path as your source cluster:

   ```bash
   sudo bash -c "echo '/dev/xvdf /vertica/backup ext4 defaults 0 0' > /etc/fstab"
   sudo mkdir /vertica/backup
   sudo mount /vertica/backup
   ```

3. Verify the success of your mounting operation by checking for data in your backup folder:

   ```bash
   ls /vertica/backup/
   ```

4. Using your admintools.conf file backup as a reference, create an empty database on the target cluster with the same dbadmin username, password, data path, and database name as your source database.

5. Stop the database, if it is running.

6. Run a restore operation:

   ```bash
   /opt/vertica/bin/vbr --config-file vmart_backup.ini --task restore
   ```

7. Start the database to conclude the restoration process.
Hard-Link Backup on AWS RAID-0

The way you perform a hard-link backup on AWS differs from doing a hard-link backup procedure on traditional bare metal Vertica installations. The use of a software RAID-0 device requires a different approach for AWS. Very small timing differences occur when taking the snapshot of the EBS volumes that make up RAID-0 devices. These timing differences can cause inconsistencies that make the backup invalid.

**Important:** Before performing a hard-link backup, you must freeze or unmount the RAID-0 file system.

You can choose between two options for performing a hard-link backup:

- Stop the cluster with admintools.
- Freeze the cluster with the `fsfreeze` command.

**Note:** You cannot perform a hard-link backup on AWS RAID-0 if your installation uses ephemeral volumes.

Hard-Link Backup with admintools

One way you can perform a hard-link backup is to stop the cluster with admintools. If you backup the volumes when the database is down, you can restore them without requiring you to run the vbr backup and restore script. However, if you want to maintain multiple point-in-time backups, you can still use vbr.

Use this approach if your service-level agreements allow you to stop the database for a period of time long enough to initiate snapshots.
Perform Hard-Link Backup Using admintools

Before performing this task, identify the instance IP addresses in your source cluster and make note of your RAID volumes for later use. Then, create a backup configuration file with hard-link backup enabled.

For information about enabling hard-link backup within a backup configuration file, see Configuring the Hard Link Local Parameter in the full Vertica documentation.
1. Stop the database:
   `admintools -t stop_db -d VMart`

2. On each node, unmount your data volume:
   `umount /vertica/data`

3. Create a snapshot of the RAID-0 volumes across your cluster, and make note of the each volume's corresponding snapshot ID. You need this information to assign snapshots to their correct node and volume designation during the restore process.

4. On each node, mount your data volume:
   `mount /vertica/data`

5. Start your database:
   `admintools -t start_db -d VMart`

6. Save the following:
   - Your RAID-0 configuration file for each node:
     `/etc/mdadm.conf`
   - Your backup configuration file:
     `vmart_backup.ini`
   - [Optional] Your admintools configuration file, which has information on your node IP addresses and mapping:
     `/opt/vertica/conf/admintools.conf`

### Hard-Link Backup with the fsfreeze Command

Another way you can perform a hard-link backup is freezing the cluster with the fsfreeze command. Because you can perform this kind of backup without stopping the cluster, users experience minimal performance effects. You must use vbr to restore from a backup performed using the fsfreeze command.
Perform a Hard-Link Backup with the fsfreeze Command

Before you can perform this procedure, you must identify the instance IP addresses in your source cluster, and make note of your RAID volumes. You also must create a backup configuration file with hard-link backup enabled. For information about enabling hard-link backup within a backup configuration file, see Configuring the Hard Link Local Parameter in the full Vertica a documentation.

1. Make a hard link backup on the RAID-0 device.
   
   ```bash
   /opt/vertica/bin/vbr --config-file vmart_backup.ini --task backup
   ```
2. Freeze the RAID-0 volume across the cluster for a consistent snapshot of the EBS volumes that constitute that device. Freezing halts all database/SQL operation until you unfreeze the volume.

   Important: Do not create a snapshot of a RAID-0 volume without freezing it first. Performing a snapshot without freezing your RAID volume invalidates your snapshot. Always check the return code of the fsfreeze command to ensure the device is frozen before you proceed.

   for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh $IP sudo fsfreeze --freeze /vertica/data;

3. Create a snapshot of the RAID-0 volumes across your cluster. Make note of each volume’s corresponding snapshot ID. You will need this information to assign snapshots to their correct volume designation during the restore process.

4. After the snapshot has started for all EBS volumes on all nodes, unfreeze the file system:

   for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh $IP sudo fsfreeze --unfreeze /vertica/data;

   Note: You do not need to wait for the snapshot to complete before unfreezing.

5. Save the following:

   - Your RAID-0 configuration file for each node:
     /etc/mdadm.conf

   - Your backup configuration file:
     vmart_backup.ini

   - [Optional] Your admintools configuration file, which has information on your node IP addresses and mapping:
     /opt/vertica/conf/admintools.conf

 Restore from a Hard-Link Backup

If a cluster fails, use your backup configuration file and backup volume snapshots to restore your cluster from your last hard-link backup.

Related Topics

The process of restoring from a hard-link backup on AWS requires the following tasks:
Create a Target Cluster

Create a target cluster on AWS. Your new instances must match your source cluster in the following ways:

- **Network and VPC**—The target instances must be in the same network and VPC as each other. However, they do not have to be in the same network and VPC as your source cluster.

- **Number of instances and nodes**—You must use the same number of instances and nodes as your source cluster.

- **IP addresses**—The internal IP addresses of the target nodes must be the same as the internal IP addresses of the nodes in the source cluster.

- **Version compatibility**—Your target cluster must be running the same AMI and Vertica versions and using the same hotfix version as your source cluster.

Your new instances may differ from the source cluster in the following ways:

- Your target cluster may be in a different availability zone.

- You may use a different instance type for your target cluster.

Restore a Database from a Hard-Link Backup

1. Create EBS volumes from your backup snapshots.

   **Note:** Before you attach your new volumes, stop and detach any existing RAID volumes mounted to the target cluster.

2. Attach the new EBS volumes to their respective nodes and volumes on the target cluster.

   **Important:** You must use the correct device mapping. For example, a backup snapshot taken for node 1, volume /dev/xvdf must be re-created in the new cluster at node 1, volume /dev/xvdf.
3. Rebuild your RAID-0 device by restoring your RAID configuration file:
   /etc/mdadm.conf

4. When the volumes have finished attaching, remount the RAID:
   for IP in 10.0.10.13 10.0.10.14 10.0.10.15; do ssh $IP sudo mount /vertica/data ;

5. Create an empty database on the target cluster with the same dbadmin username, password, datapath, and database name as your source database.

6. Run a restore operation:
   /opt/vertica/bin/vbr --config-file vmart_backup.ini --task restore

**Troubleshooting Vertica On AWS**

The following section contains information for troubleshooting Vertica on AWS.

**Checking Open Ports Manually Using the Netcat Utility**

Once your cluster is up and running, you can check ports manually through the command line using the netcat (nc) utility. What follows is an example using the utility to check ports.

Before performing the procedure, choose the private IP addresses of two nodes in your cluster.

The examples given below use nodes with the private IPs:

| 10.0.11.60 | 10.0.11.61 |

Install the nc utility on your nodes. Once installed, you can issue commands to check the ports on one node from another node.

1. To check a TCP port:

   a. Put one node in listen mode and specify the port. In the following sample, we’re putting IP 10.0.11.60 into listen mode for port 4804.

   ```
   [root@ip-10-0-11-60 ~]# nc -l 4804
   ```
b. From the other node, run `nc` specifying the IP address of the node you just put in listen mode, and the same port number.

```
[root@ip-10-0-11-61 ~]# nc 10.0.11.60 4804
```

c. Enter sample text from either node and it should show up on the other. To cancel once you have checked a port, enter Ctrl+C.

**Note:** To check a UDP port, use the same `nc` commands with the `–u` option.

```
[root@ip-10-0-11-60 ~]# nc -u -l 4804
[root@ip-10-0-11-61 ~]# nc -u 10.0.11.60 4804
```
Vertica on Microsoft Azure

Welcome to the Vertica on Microsoft Azure guide. This section explains how you can create Vertica clusters on the Microsoft Azure Cloud Platform.

Overview of Vertica on Microsoft Azure

Vertica clusters on Microsoft Azure operate on virtual machines (VMs) within a virtual network. The instructions in this document apply to VMs in Azure that are built with Vertica.

For more information about Azure VMs, see the Azure documentation.

Recommended VM Types

Vertica supports a range of Microsoft Azure virtual machine (VM) types, each optimized for different purposes. Choose the VM type that best matches your performance and price needs as a user.

For the best performance in most common scenarios, use one of the following VMs:

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory optimized</td>
<td>DS13_v2</td>
</tr>
<tr>
<td></td>
<td>DS14_v2</td>
</tr>
<tr>
<td></td>
<td>DS15_v2</td>
</tr>
<tr>
<td></td>
<td>D8s_v3</td>
</tr>
<tr>
<td></td>
<td>D16s_v3</td>
</tr>
<tr>
<td></td>
<td>D32s_v3</td>
</tr>
<tr>
<td>High memory and I/O throughput</td>
<td>GS3</td>
</tr>
<tr>
<td></td>
<td>GS4</td>
</tr>
<tr>
<td></td>
<td>GS5</td>
</tr>
<tr>
<td></td>
<td>E8s_v3</td>
</tr>
<tr>
<td></td>
<td>E16s_v3</td>
</tr>
</tbody>
</table>

Note: The GS VMs are not available in
all regions. Nor are they available on the Azure Marketplace, but you can use them by following the manual deployment steps described in Deploy Vertica on Azure: Manual Steps.

<table>
<thead>
<tr>
<th>E32s_v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>L8s</td>
</tr>
<tr>
<td>L16s</td>
</tr>
<tr>
<td>L32s</td>
</tr>
</tbody>
</table>

Note: Data stored on the drives of a VM exists only while that VM is powered on. After powering off a VM, data on temporary drives is lost.

**Supported Azure Operating Systems**

For best performance, use one of the following operating systems when deploying Vertica on Azure:

- Red Hat 7.3
- CentOS 7.3. The Azure Marketplace solution as of this writing (June 2017) is based on CentOS 7.3.1611.

For more information, see Vertica Supported Platforms.

**Deploy Vertica from the Microsoft Azure Marketplace**

For customers looking to quickly deploy a Vertica cluster in the Microsoft Azure Cloud, this section describes an automated deployment solution that can be accessed from the Azure Marketplace at Vertica Analytics Platform.

To deploy Vertica on Microsoft Azure, you need to upgrade from the Free Trial, which limits you to a 4-core quota, to Microsoft Azure’s Pay-As-You-Go subscription. The Pay-As-You-Go
subscription requires a minimum quota of 12 cores which is what is needed for the Vertica Marketplace solution for Azure.

Check with Microsoft about the benefits and credits that you can carry over from the Free Trial to the Pay-As-You-Go subscription. For more information about upgrading a free Microsoft Azure trial account to Pay-As-You-Go, see Azure pricing.

When you're ready, click Get It Now on Vertica Analytics Platform. The following sections describe the steps you need to take after you click that button.

Basic Information that Azure Needs

On the first plane (window) of the installer for your Vertica-Azure solution, enter the following information:

- **DBADMIN username**: The primary username for the virtual machines being created. This name is also used as the Vertica DBADMIN account.

- **Authentication Type**: VMs in Azure support either password or SSH public key authentication.

- **Password or SSH public key**: Enter either a password or a public key to be associated with the DBADMIN user.

- **Subscription**: Select an Azure subscription for the resources to be billed against.

- **Resource group**: Create a new resource group or select an existing resource group as the location to save all the Azure resources for your Vertica cluster.

- **Location**: Select an Azure data center to deploy the Vertica cluster. Make sure that you have enough quota assigned to that location.

Infrastructure Settings

In the second plane of the installer, make choices relating to the VMs. You need to choose two types of VMs, one to be used for the Vertica Management Console, and one to be used for the Vertica cluster nodes.

- **Management Console VM size**: Each Vertica deployment from the Azure marketplace includes a Vertica Management Console VM. This console is the jumping-off point to your
cluster. During installation, the cluster configuration is automatically imported into the MC.

- **Number of Vertica cluster nodes:** This drop-down field allows you to choose the number of nodes for Azure to deploy for the cluster. The Marketplace solution allows you to create between 1 and 16 nodes. After the nodes have been created, the installer automatically forms your Vertica cluster, configures it, and creates your database. The Community Edition license is automatically applied to the cluster, and is limited to 1 TB of RAW data, and up to 3 Vertica nodes. For deployments where more than 3 Vertica Cluster nodes are chosen, the initial database is created on the first 3 nodes. To take advantage of deployments greater than 3 nodes, customers need to obtain and apply an Enterprise license.

  **Important:** To maintain K-safety for your database, select at least three nodes.

- **Virtual machine size:** Vertica engineers have populated this selection with different VM types that are suitable for Vertica clusters. This list only includes VM types that are available in the selected location in the first plane. When you highlight this field, Azure displays those types and their specific characteristics.

- **Total RAW storage per node:** Each VM comes configured with a set of premium data disks for Vertica to use. This set is configured and presented as a single storage location. The choices in this drop-down are 768 GB, 3 TB, 6 TB, or 12 TB.

## Network Options

The third installer plane contains the networking requirements for the Vertica cluster installation. These options are:

- **Virtual network:** The Azure Marketplace solution for Vertica allows you to create a new virtual network, or select an existing virtual network, in which to place the Vertica cluster. This control on the deployment provides a point-and-click navigation to do just that.

- **Subnets:** Based on the selection of the virtual network, the Subnets entry allows you to select or create the subnet you want to use within the virtual network.

- **Public IP address—**Each VM is configured with a publicly accessible IP address. This field allows you to specify the resource name for those IP addresses, and whether they are static or dynamic. When created, the first public IP address resource is created exactly as entered, and associated with the Vertica Management Console. Azure then appends a number from
1 to 16 to the resource name for each Vertica cluster node created. This number designates which VM that resource is associated with.

- Domain name label: Because each VM has a public IP address, each node is also required to have a DNS name. Enter a prefix for the name, such as vertnode. When created, the first DNS name is created exactly as entered and associated with the Vertica Management Console. Azure then appends a number from 1 to 16 to the DNS name. That number designates which VM it is associated with. Azure adds the remaining part of the fully qualified domain name based on the location where you created the cluster.

Validate Your Configuration

Review the settings on the installer Summary plane. Azure uses this plane to validate the values you entered. In addition, Azure checks your settings against the existing quota to make sure that your solution deploys correctly.

After you are satisfied with your choices and Azure has notified you that the validation passed, click OK.

Finalize the Purchase

On the final plane of the installer, agree to the Microsoft Terms of Use, and acknowledge that you agree to pay for the resources being created.

In addition, on this plane, review the Vertica Terms of Use and the privacy policy. After you click Purchase, the deployment begins.

Monitor the Deployment

After the deployment begins, you receive a notification that the deployment is in progress. When you click the notification, the Azure portal opens the deployment status page, which displays the resources for the cluster being created.

Access the Cluster After Deployment

After the deployment successfully completes, you receive a notification in the Azure portal. When you click the notification, the Azure portal opens the resource group and displays information about all the resources created.
Finalize the Cluster Installation

Take the following steps to finalize the cluster installation:

1. Open the resource group and locate the public IP address resource for the Vertica Management Console. The public IP address resource for the Vertica Management Console is the only public IP resource without a trailing number.

2. Copy the DNS name displayed in the public IP address resource for the Vertica Management Console.

3. From a browser, enter

   https://<DNS_Name_of_VMC>:5450

   where <DNS_Name_of_VMC> is the copied value of the DNS name in the public IP address of the Vertica Management Console.

4. The browser opens the login page for the Vertica Management Console for your cluster. Log on using the default username (mcadmin) and the default password (password).

5. Accept the End User License Agreement.

6. To open the initial database, click the database icon labeled vdb on the lower left-hand section of the Vertica Management Console.

For complete information about how to use the Vertica Management Console, see Using Management Console in the Vertica documentation.

Access the Cluster VMs after Deployment

To access the cluster VMs directly, take the following steps:

1. Open the resource group and locate the public IP address resource for any of the Vertica cluster VMs. The public IP address resource for the Vertica cluster VMs is the public IP resource with a trailing number after the resource name. The trailing number corresponds to the VM number. For example, PublicIP1 maps to Vertica cluster node 1, and so on.

2. Copy the DNS name displayed in the public IP address resource for the Vertica cluster node.
3. Connect to the node using the dbadmin user you created. Log in with the authentication method you chose (public key or password) using an SSH connection tool.

At this point, your cluster is fully installed and you are ready to use your new database.

**Deploy Vertica on Azure: Manual Steps**

To start creating your Vertica cluster in Azure using manual steps, you first need to create a VM. During the VM creation process, you create and configure the other resources required for your cluster, which are then available for any additional VMs that you create.

Follow these procedures to deploy Vertica on Azure.

**Configuring and Launching a New Instance**

An Azure VM is similar to a traditional host. Just as with an on-premises cluster, you must prepare and configure the hardware settings for your cluster and network before you install Vertica.

The first steps are:

1. From the Azure marketplace, select an operating system that Vertica supports.
2. Select a VM type. See Recommended VM Types
3. Choose a deployment model. For best results, choose the resource manager deployment model.

**Configure Network Security Group**

Vertica has specific network security group requirements, as described in Network Security Group Configurations.

Create and name your own network security group, following these guidelines.

You must configure SSH as:

- Protocol: TCP
- Source port range: Any
- Destination port range: 22
- Source: Any
- Destination: Any

You can make additional modifications, based on your specific requirements.

**Add Disk Containers**

Create an Azure storage account, which later contains your cluster storage disk containers. For optimal throughput, select Premium storage and align the storage to a GS or D Sv2 VM.

For more information about what a storage account is, and how to create one, refer to [About Azure storage accounts](#).

**Configure Credentials**

Create a password or assign an SSH key pair to use with Vertica.

For information about how to use key pairs in Azure, see [How to create and use an SSH public and private key pair for Linux VMs in Azure](#).

**Assign a Public IP Address**

A public IP is an IP address that you can use to connect to your cluster externally. For best results, assign a single static public IP to a node in your cluster. You can then connect to other nodes in your cluster from your primary node using the internal IP addresses that Azure generated when you specified your virtual network settings.

By default, a public IP address is dynamic; it changes every time you shut down the server. You can choose a static IP address, but doing so can add cost to your deployment.

During a VM installation, you cannot set a DNS name. If you use dynamic public IPs, set the DNS name in the public IP resource for each VM after deployment.

For information about public IP addresses, refer to [IP address types and allocation methods in Azure](#).

**Create Additional VMs**

If needed, to create additional VMs, repeat the previous instructions in this document.
Connect to a Virtual Machine

Before you can connect to any of the VMs you created, you must first make your virtual network externally accessible. To do so, you must attach the public IP address you created during network configuration to one of your VMs.

Connect to Your VM

To connect to your VM, complete the following tasks:

1. Connect to your VM using SSH with the public IP address you created in the configuration steps.

2. Authenticate using the credentials and authentication method you specified during the VM creation process.

Connect to Other VMs

Connect to other virtual machines in your virtual network by first using SSH to connect to your publicly connected VM. Then, use SSH again from that VM to connect through the private IP addresses of your other VMs.

If you are using private key authentication, you may need to move your key file to the root directory of your publicly connected VM. Then, use PuTTY or WinSCP to connect to other VMs in your virtual network.

Prepare the Virtual Machines

After you create your VMs, you need to prepare them for cluster formation.

Add the Vertica License and Private Key

Prepare your nodes by adding your private key (if you are using one) to each node and to your Vertica license. These steps assume that the initial user you configured is the DBADMIN user.
1. As the DBADMIN user, copy your private key file from where you saved it locally onto your primary node.

   Depending upon the procedure you use to copy the file, the permissions on the file may change. If permissions change, the install_vertica script fails with a message similar to the following:

   ```
   Failed Login Validation 10.0.2.158, cannot resolve or connect to host as root.
   ```

   If you receive a failure message, enter the following command to correct permissions on your private key file:

   ```bash
   $ chmod 600 /<name-of-key>.pem
   ```

2. Copy your Vertica license to your primary VM. Save it in your home directory or other known location.

**Install Software Dependencies for Vertica on Azure**

In addition to the Vertica standard Package Dependencies, as the root user, you must install the following packages before you install Vertica on Azure:

- `pstack`
- `mcelog`
- `sysstat`
- `dialog` (Required for Vertica 8.0.x)

**Configure Storage**

Use a dedicated Azure storage account for node storage.

**Caution:** Do *not* store your information on the root volume, especially your data and catalog directories. Storing information on the root volume may result in data loss.

When configuring your storage, make sure to use a supported file system.
Attach Disk Containers to Virtual Machines (VMs)

Using your previously created storage account, attach disk containers to your VMs that are appropriate to your needs.

For best performance, combine multiple storage volumes into RAID-0. For most RAID-0 implementations, attach 6 storage disk containers per VM.

Combine Disk Containers for Storage

If you are using RAID, follow these steps to create a RAID-0 drive on your VMs. The following example shows how you can create a RAID-0 volume named md10 from 6 individual volumes named:

- sdc
- sdd
- sde
- sdf
- sdg
- sdh

1. Form a RAID-0 volume using the mdadm utility:

   ```
   $ mdadm --create /dev/md10 --level 0 --raid-devices=6
   /dev/sdc /dev/sdd /dev/sde /dev/sdf /dev/sdg /dev/sdh
   ```

2. Format the file system to be one that Vertica supports:

   ```
   $ mkfs.ext4 /dev/md10
   ```

3. Find the UUID on the newly formed RAID volume:

   ```
   ls -l /dev/disk/by-uuid
   ```

4. Copy the UUID and place it in the FSTAB file.

   ```
   $ sudo vi /etc/fstab
   UUID=<uuid here> /dev/md10 ext4 defaults , errors=remount-ro 0 1
   ```
5. Create folders for your Vertica data and catalog.

   $ mkdir /vertica
   $ mkdir /vertica/data

6. Mount the RAID volume to your data and catalog directories.

   $ mount /dev/md10 /home/dbadmin/vertica/data

Create a Swap File

In addition to storage volumes to store your data, Vertica requires a swap volume or swap file to operate.

Create a swap file or swap volume of at least 2 GB. The following steps show how to create a swap file within Vertica on Azure:

1. Install devnull and swapfile:

   $ install -o root -g root -m 0600 /dev/null /swapfile

2. Create the swap file:

   $ dd if=/dev/zero of=/swapfile bs=1024 count=2048k

3. Prepare the swap file using mkswap:

   $ mkswap /swapfile

4. Use swapon to instruct Linux to swap on the swap file:

   $ swapon /swapfile

5. Persist the swapfile in FSTAB:

   $ echo "/swapfile swap swap auto 0 0" >> /etc/fstab

Repeat the volume attachment, combination, and swap file creation procedures for each VM in your cluster.
For More Information

- About Azure storage accounts
- Prepare Disk Storage Locations

Download Vertica

To download the Vertica server appropriate for your operating system and license type, go to my.vertica.com/download/vertica.

Run the rpm to extract the files.

After you complete the download and extraction, the next section describes how to use the install_vertica script to form a cluster and install the Vertica database software.

Form a Cluster and Install Vertica

Use the install_vertica script to combine two or more individual VMs to form a cluster and install the Vertica database.

Before You Start

Before you run the install_vertica script:

- Check the Virtual Network page for a list of current VMs and their associated private IP addresses.

- Identify your storage location. The installer assumes that you have mounted your storage to /vertica/data. To specify another location, use the --data-dir argument.

- Identify your storage location. To create your database's data directory on mounted RAID drive, when you run the install_vertica script, provide /vertica/data as the value of the --data-dir option.

Caution: Do not store your data on the root drive.
Combine Virtual Machines (VMs)

The following example shows how to combine VMs using the `install_vertica` script.

1. While connected to your primary node, construct the following command to combine your nodes into a cluster.

   ```bash
   $ sudo /opt/vertica/sbin/install_vertica --hosts 10.2.0.164,10.2.0.165,10.2.0.166 --dba-user-password-disabled --point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-private-key>.pem --license <license.file>
   ```

2. Substitute the IP addresses for your VMs and include your root key file name, if applicable.

3. Include the `--point-to-point` parameter to configure spread to use direct point-to-point communication between all Vertica nodes, as required for clusters on Azure when installing or updating Vertica.

4. If you are using Vertica Community Edition, which limits you to three nodes, specify `-L CE` with no license file.

5. After you combine your nodes, to reduce security risks, keep your key file in a secure place—separate from your cluster—and delete your on-cluster key with the shred command:

   ```bash
   $ shred examplekey.pem
   ```

   **Important:** You need your key file to perform future Vertica updates.

6. Reboot your cluster to complete the cluster formation and Vertica installation.

For complete information on the `install_vertica` script and its parameters, see [Installing Vertica with the Installation Script](#).

### After Your Cluster is Up and Running

Now that your cluster is configured and running, and Vertica is running, take these steps:
1. **Creating a Database.** When you installed Vertica, a database administrator user was created: DBADMIN.

2. Use this account to create and start a database. For detailed information about the DBADMIN role, see DBADMIN Role.

3. **Configuring the Database.** After the database is installed, it's important to configure its settings, schemas, and security.

## Using Management Console (MC) on Azure

The Vertica solution in the Azure marketplace provides a Management Console (MC) virtual machine. The MC VM allows you to deploy a VM dedicated to running MC onto a Vertica cluster on Azure.

You can deploy an MC VM to your Vertica on Azure cluster at any time, as long as your cluster meets the following requirements:

- Your Azure cluster must be running Vertica version 8.0.x or later.
- Your MC version must be compatible with the Vertica version of your cluster.

## Deploy an MC VM

To deploy MC onto your target Vertica cluster on Azure, perform these tasks:

1. Select the Vertica Management Console deployment from the Azure Marketplace.

2. Specify an admin username for your new MC VM.

3. Place your MC VM in the same location as your target cluster.

4. Place your MC VM in a resource group, separate from your target cluster.

5. Specify the virtual network and subnet of your target cluster.

6. Specify a DNS name and public IP address.
Configure the Network Security Group for the MC VM

...The MC VM requires specific security group configurations in order to communicate with the Vertica cluster. Configure the security rules for the MC VM as described in Configuring and Launching a New Instance.

Log in to MC and Manage Your Cluster

After you have deployed your MC VM and configured your security group settings, Enter the following into your browser navigation field. This command launches the Management Console:

```
https://[IP address or DNS name]:5450
```

Log in to Management Console using the public IP address that you specified during instance creation.

From there, you can manage your Vertica database on Azure using standard MC procedures.

Uninstalling MC

To uninstall MC on Azure, follow the procedures provided in Uninstalling Management Console before terminating the MC node.

Related Topics

- Using Management Console
- Managing Database Clusters

Deploy the Vertica Test Drive for Azure

For users looking to experience Vertica in a “test drive” environment on Azure, we offer that capability at the Azure Marketplace at Vertica Analytics Platform.

When you click the Test Drive button, Azure creates a single Vertica node that is preloaded with clickstream and sales data. This Test Drive allows you to explore and analyze the data before and after conducting A/B testing.
After the deployment completes, you receive an email with instructions on how to access the Test Drive. You can also consult the Test Drive user manual: Vertica Test Drive for Clickstream Analytics.
Vertica on Google Cloud Platform

Welcome to the Vertica on Google Cloud Platform guide.

Vertica clusters on Google Cloud Platform (GCP) operate on top of virtual machines (VMs) as part of the Google Compute Engine interconnected using a virtual private cloud (VPC) network. The VMs are accessible using the Internet with standard SSH connections. To access the Vertica Management Console, use HTTPS on port 5450.

The following topics describe several deployment methods to run Vertica on Google Cloud Platform:

Overview of Vertica on Google Cloud Platform

Vertica clusters on Google Cloud Platform (GCP) operate on top of virtual machines (VMs) as part of the Google Compute Engine interconnected using a virtual private cloud (VPC) network. The VMs are accessible using the Internet with standard SSH connections. To access the Vertica Management Console, use HTTPS on port 5450.

The following topics describe several deployment methods to run Vertica on Google Cloud Platform:

- Deploy Vertica from the Google Cloud Launcher Marketplace
- Deploy Vertica on Google Cloud Platform: Manual Steps

Deploy Vertica from the Google Cloud Launcher Marketplace

Are you looking to quickly deploy a Vertica cluster in the Google Cloud Platform (GCP)? If so, this section describes a Vertica-developed automated solution that you can access from the Vertica Analytics Platform Google Cloud Launcher.

The Vertica Cloud Launcher solution allows you to create up a 16-node cluster. The solution includes the Vertica Management Console (MC) as the primary UI for you to get started.

Each deployment automatically creates an initial database called vdb using the Community Edition (CE) license. The CE license is limited to a maximum of 3 nodes. In cases where you
want to use more than 3 nodes for deployment on GCP, the deployment creates the initial database on the first 3 nodes in the cluster.

After the launcher creates the initial database (vdb), it preconfigures the MC to attach to that database automatically.

Configure the Vertica Cloud Launcher Solution

To get started with a deployment of Vertica from the Google Cloud Launcher, go to Vertica Analytics Platform Google Cloud Launcher.

Follow these steps:

1. From the listing page, to start the solution launcher, click LAUNCH ON COMPUTE ENGINE.

2. Enter the following information:

   - **Deployment name**—Each deployment must have a unique name. That name is used as the prefix for the names of all VMs created during the deployment. The deployment name can only contain lowercase characters, numbers, and dashes. The name must start with a lowercase letter and cannot end with a dash.

   - **Zone**—GCP breaks its cloud data centers into regions and zones. *Regions* are a collection of zones in the same geographical location. *Zones* are collections of compute resources, which vary from zone to zone.

     For best results, pick the zone in your designated region that supports the latest Intel CPUs. For a complete listing of regions and zones, including supported processors, see Regions and Zones.

   - **Machine type for Vertica Management Console**—The Vertica Analytics Platform in Cloud Launcher always deploys the Vertica Management Console (MC) as part of the solution.

     The default machine type for MC is the n1-standard-4. However, you can choose another machine type that better suits the needs requires by the MC node for additional purposes, such serving as a target node for backups, data transformation, or additional management tools.

   - **Node count for Vertica Cluster**—The total number of VMs you want to deploy in the Vertica Cluster. The default is 3.
Note: The Cloud Launcher automatically deploys the Vertica Community Edition license, which limits the database to 3 nodes and up to 1 TB in raw data. If you intend to use the CE license, deployment of 4 or more nodes will result in an overpopulated cluster where VM resources are deployed (and incur costs) without being used by the database.

- **Machine type for Vertica Cluster nodes**—The Cloud Launcher builds each node in the cluster using the same machine type. The default value is the n1-highmem-16 type. Modify the machine type for your nodes based on the needs and costs associated with your workloads.

- **Data disk type**—GCP offers two types of persistent disk storage: Standard and SSD. The costs associated with Standard are less, but the performance of SSD storage is much better. Vertica recommends you use SSD storage. For more information on Standard and SSD persistent disks, see Storage Options.

- **Disk size for the Data disk**—Disk performance is directly tied to the disk size in GCP. The default value of 2000 GBs (2 TB) is the minimum disk size for SSD persistent disks that allows maximum throughput. If you select a smaller disk size, the throughput performance decreases. If you select a large disk size, the performance remains the same as the 2 TB option.

- **Network name**—VMs in GCP must exist on a virtual private cloud (VPC). When you created your GCP account, a default VPC was created. Create additional VPCs to isolate solutions or projects from one another. The Vertica Analytics Platform creates all the nodes in the same VPC.

- **Subnetwork name**—Just as a GCP account may have multiple VPCs, each VPC may also have multiple subnets. Use additional subnets to group or isolate solutions within the same VPC.

After you have entered all the required information, click Deploy to begin the deployment process.

**Monitor the Deployment**

After the deployment begins, Google Cloud Launcher automatically opens a page that displays the status of the resources being created. Items that are still being processed have a spinning circle to the left of them and the text is a light gray color. Items that have been created are dark gray in color, with an icon designating that resource type on the left.
After the deployment completes, a green check mark appears next to the deployment name in the upper left-hand section of the screen.

**Accessing the Cluster After Deployment**

After the deployment completes, the right-hand section of the screen displays the following information:

- **dbadmin password**—A randomly generated password for the dbadmin account on the nodes. For security reasons, change the dbadmin password when you first log in to one of the Vertica cluster nodes.

- **mcadmin password**—A randomly generated password for the mcadmin account for accessing the Management Console. For security reasons, change the mcadmin password after you first log in to the MC.

- **Vertica Node 1 IP address**—The external IP address for the first node in the Vertica cluster is exposed here so that you can connect to the VM using a standard SSH client. To access the MC, press the Access Vertica MC button in the Get Started section of the dialog box. Copy the mcadmin password and paste it when asked.

For more information on using the MC, click **Using Management Console**.

**Access the Cluster Nodes**

There are two ways to access the cluster nodes directly:

- **Use GCP’s integrated SSH shell** by selecting the SSH button in the Get Started section. This shell opens a pop-up in your browser that runs GCP’s web-based SSH client. You are automatically logged on as the user you authenticated as in the GCP environment.

  After you have access to the first Vertica cluster node, execute the `su dbadmin` command, and authenticate using the dbadmin password.

- **In addition, use other standard SSH clients** to connect directly to the first Vertica cluster node. Use the Vertica Node 1 IP address listed on the screen as the dbadmin user, and authenticate with the dbadmin password.

  Follow the on-screen directions to log in using the mcadmin account and accept the EULA. After you've been authenticated, access the initial database by clicking the vdb icon (looks like a green cylinder) in the Recent Databases section.
When the cluster is fully deployed, check out the Getting Started Guide of the Vertica documentation.

**Deploy Vertica on Google Cloud Platform: Manual Steps**

Before you create your Vertica cluster in Google Cloud Platform (GCP) using manual steps, you must create a virtual machine (VM) instance from the Compute Engine section of GCP.

**Configure and Launch a New Instance**

All VM instances that you create should be launched in the same virtual public cloud (VPC).

To configure and launch a new VM instance, follow these instructions:

1. From within the Compute Engine section of GCP, from the menu on the left-hand site of the screen, select VM Instances.
   
   GCP displays all the VM instances that you have created so far.

2. Select the CREATE INSTANCE link.

3. Enter a name for the new instance.

4. Select the zone where you plan to deploy the instance.

   GCP breaks its cloud data centers down by regions and zones. *Regions* are a collection of zones that are all in the same geographical location. Zones are collections of compute resources, which vary from zone to zone. Always pick the zone in your designated region that supports the latest Intel CPUs.

   For a complete listing of regions and zones, including supported processors, see Regions and Zones.

5. Select a machine type.

   GCE offers many different types of VM instances. For best results, only deploy Vertica on VM instances with 8 vCPUs or more and at least 30 GB of RAM.
6. Select the boot disk (image).

You create VM instances from a public or custom image. If you are starting with Vertica in GCP for the first time, select either the CentOS 7 or RHEL 7 public image. Those images have been tested thoroughly with Vertica.

For more information about deploying a VM instance, see Creating and Starting an Instance.

After you have configured the VM instance to be used as a Vertica cluster node, GCP allows you to convert that instance into a custom image. Doing so allows you to deploy multiple versions of that VM instance; each VM instance is identical except for the node name and IP address.

For more information about creating a custom image, see Creating, Deleting, and Deprecating Custom Images.

Connect to a Virtual Machine

Before you can connect to any of the VMs you created, you must first identify the external IP address. The VM instance section of GCP contains a list of all currently deployed VMs and their associated external IP addresses.

Connect to Your VM

To connect to your VM, complete the following tasks:

1. Connect to your VM using SSH with the external IP address you created in the configuration steps.

2. Authenticate using the credentials and SSH key that you provided to your GCP account upon creation.

Connect to Other VMs

To connect to other virtual machines in your virtual network:
1. Use SSH to connect to your publicly connected VM.

2. Use SSH again from that VM to connect through the private IP addresses of your other VMs.

Because GCP forces the use of private key authentication, you may need to move your key file to the root directory of your publicly connected VM. Then, use SSH to connect to other VMs in your virtual network.

**Prepare the Virtual Machines**

After you create your VMs, you need to prepare them for cluster formation.

**Add the Vertica License and Private Key**

Prepare your nodes by adding your private key (if you are using one) to each node and to your Vertica license. The following steps assume that the initial user you configured is the DBADMIN user:

1. As the DBADMIN user, copy your private key file from where you saved it locally onto your primary node.

   Depending upon the procedure you use to copy the file, the permissions on the file may change. If permissions change, the install_vertica script fails with a message similar to the following:

   ```
   Failed Login Validation 10.0.2.158, cannot resolve or connect to host as root.
   ```

   If you see the previous failure message, enter the following command to correct permissions on your private key file:

   ```
   $ chmod 600 /<name-of-key>.pem
   ```

2. Copy your Vertica license to your primary VM. Save it in your home directory or other known location.
Install Software Dependencies for Vertica on GCP

In addition to the Vertica standard package dependencies, as the root user, you must install the following packages before you install Vertica:

- `pstack`
- `mcelog`
- `sysstat`
- `dialog` (Required for Vertica 8.0.x and later)

Configure Storage

For best disk performance in GCP, Vertica recommends customers use SSD persistent storage, configured to at least 2TB (2000 GB) in size. Disk performance is directly tied to the disk size in GCP. 2000 GBs (2TB) is the minimum disk size for SSD persistent disks that allows maximum throughput.

**Caution:** Do not store your information on the root volume, especially in your data and catalog directories. Storing information on the root volume may result in data loss.

When configuring your storage, make sure to use a supported file system.

Create a Swap File

In addition to storage volumes to store your data, Vertica requires a swap volume or swap file for the setup script to complete.

Create a swap file or swap volume of at least 2 GB. The following steps show how to create a swap file within Vertica on GCP:

1. Install the `devnull` and `swapfile` files:

   ```
   $ install -o root -g root -m 0600 /dev/null /swapfile
   ```

2. Create the swap file:
3. Prepare the swap file using `mkswap`:

   ```bash
   $ mkswap /swapfile
   ```

4. Use `swapon` to instruct Linux to swap on the swap file:

   ```bash
   $ swapon /swapfile
   ```

5. Persist the swapfile in FSTAB:

   ```bash
   $ echo "/swapfile swap swap auto 0 0" >> /etc/fstab
   ```

6. Repeat the volume attachment, combination, and swap file creation procedures for each VM in your cluster.

### Download Vertica

To download the Vertica server appropriate for your operating system and license type, go to [https://my.vertica.com/download/vertica](https://my.vertica.com/download/vertica).

Download the rpm and run the rpm to extract the files as described in Download and Install the Vertica Server Package in the Vertica documentation.

After you complete the download and extraction, use the `install_vertica` script to form a cluster and install the Vertica database software, as described in the next section.

### Form a Cluster and Install Vertica

Use the `install_vertica` script to combine two or more individual VMs to form a cluster and install your Vertica database.

Before you run the `install_vertica` script, follow these steps:

1. Check the VM Instances page of the Compute Engine section on GCP to locate a list of current VMs and their associated internal IP addresses.
2. Identify your storage location on your VMs. The installer assumes that you have mounted your storage to `/home/dbadmin`. To specify another location, use the `--data-dir` argument.

**Caution:** Do not store your data on the root drive.

The following steps show how to combine virtual machines (VMs) into a cluster using the `install_vertica` script:

1. While connected to your primary node, construct the following command to combine your nodes into a cluster.

   ```
   $ sudo /opt/vertica/sbin/install_vertica --hosts 10.2.0.164,10.2.0.165,10.2.0.166 --dba-user-password-disabled --point-to-point --data-dir /vertica/data --ssh-identity ~/<name-of-private-key>.pem --license <license.file>
   ```

2. Substitute the IP addresses for your VMs, and include your root key file name, if applicable.

3. Include the `--point-to-point` parameter to configure spread to use direct point-to-point communication among all Vertica nodes, as required for clusters on GCP when installing or updating Vertica.

4. If you are using Vertica Community Edition, which limits you to three nodes, specify `-L CE` with no license file.

5. After you combine your nodes, to reduce security risks, keep your key file in a secure place—separate from your cluster—and delete your on-cluster key with the shred command:

   ```
   $ shred examplekey.pem
   ```

**Important:** You need your key file to perform future Vertica updates.

For complete information about the `install_vertica` script and its parameters, see [Installing Vertica with the Installation Script](#).

**After Your Cluster is Up and Running**

Now that your cluster is configured and running, and Vertica is running, take these steps:
1. **Create a Database.**

   When you installed Vertica, a database administrator user was created: DBADMIN. Use this account to create and start a database.

   For detailed information about the DBADMIN role, see [DBADMIN Role](#).

2. **Configure the Database.** After the database is installed, it's important to configure its settings, schemas, and security.
Integrating with Apache Hadoop

Apache™ Hadoop™, like Vertica, uses a cluster of nodes for distributed processing. The primary component of interest is HDFS, the Hadoop Distributed File System.

You can use Vertica with HDFS in several ways:

- You can import HDFS data into locally-stored ROS files.
- You can access HDFS data in place using external tables. You can define the tables yourself or get schema information from HCatalog, a Hadoop component.
- You can use HDFS as a storage location for ROS files.
- You can export data from Vertica to share with other Hadoop components using a Hadoop columnar format.

See Hadoop Interfaces for more information about these options.

A Hadoop cluster can use Kerberos authentication to protect data stored in HDFS. Vertica integrates with Kerberos to access HDFS data if needed. See Using Kerberos with Hadoop.
Hadoop Distributions

Vertica can be used with Hadoop distributions from Hortonworks, Cloudera, and MapR. See Vertica Integrations for Hadoop for the specific versions that are supported.

If you are using Cloudera, you can manage your Vertica cluster using Cloudera Manager. See Integrating With Cloudera Manager.

If you are using MapR, see Integrating Vertica with the MapR Distribution of Hadoop.
Cluster Architecture

Vertica supports two cluster architectures. Which architecture you use affects the decisions you make about integration. These options might also be limited by license terms.

- You can co-locate Vertica on some or all of your Hadoop nodes. Vertica can then take advantage of data locality.

- You can build a Vertica cluster that is separate from your Hadoop cluster. In this configuration, Vertica can fully use each of its nodes; it does not share resources with Hadoop.

These layout options are described in Cluster Layout.
File Paths

Hadoop file paths are expressed as URLs in the hdfs or webhdfs URL scheme. If you need to escape a special character in a path, use URL escaping. All input characters that are not a-z, A-Z, 0-9, '-', ',', '_' or '~' must be converted to URL encoding (%NN where NN is a two-digit hexadecimal number). The following example URL-encodes a file name with a space in it.

```
hdfs:///opt/data/my%20file.orc
```

You can use globs, including regular expressions, in file paths. When Hive writes data, it sometimes creates temporary files with a "_COPYING" suffix. If you try to read these files into Vertica you will get an error message, because they are not a valid format. The following example copies only files ending in digits, the usual format for exports from Hive:

```
=> CREATE EXTERNAL TABLE (...) AS COPY FROM hdfs:///data/parquet/*_[0-9] PARQUET;
```
Cluster Layout

Vertica and Hadoop each use a cluster of nodes for distributed processing. These clusters can be co-located, meaning you run both products on the same machines, or separate. See Co-Located Clusters and Separate Clusters.

With either architecture, if you are using the hdfs scheme to read ORC or Parquet files, you must do some additional configuration. See Configuring the hdfs Scheme.

Co-Located Clusters

With co-located clusters, Vertica is installed on some or all of your Hadoop nodes. The Vertica nodes use a private network in addition to the public network used by all Hadoop nodes, as the following figure shows:

You might choose to place Vertica on all of your Hadoop nodes or only on some of them. If you are using HDFS Storage Locations you should use at least three Vertica nodes, the minimum number for K-Safety.

Using more Vertica nodes can improve performance because the HDFS data needed by a query is more likely to be local to the node.

You can place Hadoop and Vertica clusters within a single rack, or you can span across many racks and nodes. If you do not co-locate Vertica on every node, you can improve performance by co-locating it on at least one node in each rack. See Configuring Rack Locality.

Normally, both Hadoop and Vertica use the entire node. Because this configuration uses shared nodes, you must address potential resource contention in your configuration on those nodes. See Configuring Hadoop for Co-Located Clusters for more information. No changes are needed on Hadoop-only nodes.
Hardware Recommendations

Hadoop clusters frequently do not have identical provisioning requirements or hardware configurations. However, Vertica nodes should be equivalent in size and capability, per the best-practice standards recommended in General Hardware and OS Requirements and Recommendations in Installing Vertica.

Because Hadoop cluster specifications do not always meet these standards, Vertica recommends the following specifications for Vertica nodes in your Hadoop cluster.

<table>
<thead>
<tr>
<th>Specifications For...</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>For best performance, run:</td>
</tr>
<tr>
<td></td>
<td>• Two-socket servers with 8–14 core CPUs, clocked at or above 2.6 GHz for clusters over 10 TB</td>
</tr>
<tr>
<td></td>
<td>• Single-socket servers with 8–12 cores clocked at or above 2.6 GHz for clusters under 10 TB</td>
</tr>
<tr>
<td>Memory</td>
<td>Distribute the memory appropriately across all memory channels in the server:</td>
</tr>
<tr>
<td></td>
<td>• Minimum—8 GB of memory per physical CPU core in the server</td>
</tr>
<tr>
<td></td>
<td>• High-performance applications—12–16 GB of memory per physical core</td>
</tr>
<tr>
<td></td>
<td>• Type—at least DDR3-1600, preferably DDR3-1866</td>
</tr>
<tr>
<td>Storage</td>
<td>Read/write:</td>
</tr>
<tr>
<td></td>
<td>• Minimum—40 MB/s per physical core of the CPU</td>
</tr>
<tr>
<td></td>
<td>• For best performance—60–80 MB/s per physical core</td>
</tr>
<tr>
<td></td>
<td>Storage post RAID: Each node should have 1–9 TB. For a production setting, Vertica recommends RAID 10. In some cases, RAID 50 is acceptable.</td>
</tr>
<tr>
<td></td>
<td>Because Vertica performs heavy compression and encoding, SSDs are not required. In most cases, a RAID of more, less-expensive HDDs performs just as well as a RAID of fewer SSDs.</td>
</tr>
</tbody>
</table>
If you intend to use RAID 50 for your data partition, you should keep a spare node in every rack, allowing for manual failover of a Vertica node in the case of a drive failure. A Vertica node recovery is faster than a RAID 50 rebuild. Also, be sure to never put more than 10 TB compressed on any node, to keep node recovery times at an acceptable rate.

| Network | 10 GB networking in almost every case. With the introduction of 10 GB over cat6a (Ethernet), the cost difference is minimal. |

---

**Configuring Hadoop for Co-Located Clusters**

If you are co-locating Vertica on any HDFS nodes, there are some additional configuration requirements.

**Hadoop Configuration Parameters**

For best performance, set the following parameters with the specified minimum values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS block size</td>
<td>512MB</td>
</tr>
<tr>
<td>Namenode Java Heap</td>
<td>1GB</td>
</tr>
<tr>
<td>Datanode Java Heap</td>
<td>2GB</td>
</tr>
</tbody>
</table>

**WebHDFS**

Hadoop has two services that can provide web access to HDFS:

- WebHDFS
- httpFS

For Vertica, you must use the WebHDFS service.
YARN

The YARN service is available in newer releases of Hadoop. It performs resource management for Hadoop clusters. When co-locating Vertica on YARN-managed Hadoop nodes you must make some changes in YARN.

Vertica recommends reserving at least 16GB of memory for Vertica on shared nodes. Reserving more will improve performance. How you do this depends on your Hadoop distribution:

- If you are using Hortonworks, create a "Vertica" node label and assign this to the nodes that are running Vertica.

- If you are using Cloudera, enable and configure static service pools.

Consult the documentation for your Hadoop distribution for details. Alternatively, you can disable YARN on the shared nodes.

Hadoop Balancer

The Hadoop Balancer can redistribute data blocks across HDFS. For many Hadoop services, this feature is useful. However, for Vertica this can reduce performance under some conditions.

If you are using HDFS storage locations, the Hadoop load balancer can move data away from the Vertica nodes that are operating on it, degrading performance. This behavior can also occur when reading ORC or Parquet files if Vertica is not running on all Hadoop nodes. (If you are using separate Vertica and Hadoop clusters, all Hadoop access is over the network, and the performance cost is less noticeable.)

To prevent the undesired movement of data blocks across the HDFS cluster, consider excluding Vertica nodes from rebalancing. See the Hadoop documentation to learn how to do this.

Replication Factor

By default, HDFS stores three copies of each data block. Vertica is generally set up to store two copies of each data item through K-Safety. Thus, lowering the replication factor to 2 can save space and still provide data protection.

To lower the number of copies HDFS stores, set HadoopFSReplication, as explained in Troubleshooting HDFS Storage Locations.
Disk Space for Non-HDFS Use

You also need to reserve some disk space for non-HDFS use. To reserve disk space using Ambari, set `dfs.datanode.du.reserved` to a value in the `hdfs-site.xml` configuration file.

Setting this parameter preserves space for non-HDFS files that Vertica requires.

**Configuring Rack Locality**

*Note: This feature is supported only for reading ORC and Parquet data on co-located clusters. It is only meaningful on Hadoop clusters that span multiple racks.*

When possible, when planning a query Vertica automatically uses database nodes that are co-located with the HDFS nodes that contain the data. Moving query execution closer to the data reduces network latency and can improve performance. This behavior, called node locality, requires no additional configuration.

When Vertica is co-located on only a subset of HDFS nodes, sometimes there is no database node that is co-located with the data. However, performance is usually better if a query uses a database node in the same rack. If configured with information about Hadoop rack structure, Vertica attempts to use a database node in the same rack as the data to be queried.

For example, the following diagram illustrates a Hadoop cluster with three racks each containing three data nodes. (Typical production systems have more data nodes per rack.) In each rack, Vertica is co-located on one node.
If you configure rack locality, Vertica uses db01 to query data on dn1, dn2, or dn3, and uses db02 and db03 for data on rack2 and rack3 respectively. Because HDFS replicates data, any given data block can exist in more than one rack. If a data block is replicated on dn2, dn3, and dn6, for example, Vertica uses either db01 or db02 to query it.

Hadoop components are rack-aware, so configuration files describing rack structure already exist in the Hadoop cluster. To use this information in Vertica, configure fault groups that describe this rack structure. Vertica uses fault groups in query planning.

### Configuring Fault Groups

Vertica uses Fault Groups to describe physical cluster layout. Because your database nodes are co-located on HDFS nodes, Vertica can use the information about the physical layout of the HDFS cluster.
Tip: For best results, ensure that each Hadoop rack contains at least one co-located Vertica node.

Hadoop stores its cluster-layout data in a topology mapping file in HADOOP_CONF_DIR. On HortonWorks the file is typically named topology_mappings.data. On Cloudera it is typically named topology.map. Use the data in this file to create an input file for the fault-group script. For more information about the format of this file, see Creating a Fault Group Input File.

Following is an example topology mapping file for the cluster illustrated previously:

```
[network_topology]
dn1.example.com=/rack1
10.20.41.51=/rack1
dn2.example.com=/rack1
10.20.41.52=/rack1
dn3.example.com=/rack1
10.20.41.53=/rack1
dn4.example.com=/rack2
10.20.41.71=/rack2
dn5.example.com=/rack2
10.20.41.72=/rack2
dn6.example.com=/rack2
10.20.41.73=/rack2
dn7.example.com=/rack3
10.20.41.91=/rack3
dn8.example.com=/rack3
10.20.41.92=/rack3
dn9.example.com=/rack3
10.20.41.93=/rack3
```

From this data, you can create the following input file describing the Vertica subset of this cluster:

```
/rack1 /rack2 /rack3
/rack1 = db01
/rack2 = db02
/rack3 = db03
```

This input file tells Vertica that the database node "db01" is on rack1, "db02" is on rack2, and "db03" is on rack3. In creating this file, ignore Hadoop data nodes that are not also Vertica nodes.

After you create the input file, run the fault-group tool:

```
$ python /opt/vertica/scripts/fault_group ddl_generator.py dbName input_file > fault_group_ddl.sql
```

The output of this script is a SQL file that creates the fault groups. Execute it following the instructions in Creating Fault Groups Using the Fault Group Script.

You can review the new fault groups with the following statement:
SELECT member_name, node_address, parent_name FROM fault_groups
INNER JOIN nodes ON member_name=node_name ORDER BY parent_name;

<table>
<thead>
<tr>
<th>member_name</th>
<th>node_address</th>
<th>parent_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>db01</td>
<td>10.20.41.51</td>
<td>/rack1</td>
</tr>
<tr>
<td>db02</td>
<td>10.20.41.71</td>
<td>/rack2</td>
</tr>
<tr>
<td>db03</td>
<td>10.20.41.91</td>
<td>/rack3</td>
</tr>
</tbody>
</table>

(3 rows)

Working With Multi-Level Racks

A Hadoop cluster can use multi-level racks. For example, /west/rack-w1, /west/rack-2, and /west/rack-w3 might be served from one data center, while /east/rack-e1, /east/rack-e2, and /east/rack-e3 are served from another. Use the following format for entries in the input file for the fault-group script:

/west /east
/west = /rack-w1 /rack-w2 /rack-w3
/east = /rack-e1 /rack-e2 /rack-e3
/rack-w1 = db01
/rack-w2 = db02
/rack-w3 = db03
/rack-e1 = db04
/rack-e2 = db05
/rack-e3 = db06

Do not create entries using the full rack path, such as /west/rack-w1.

Auditing Results

To see how much data can be loaded with rack locality, use EXPLAIN with the query and look for statements like the following in the output:

100% of ORC data including co-located data can be loaded with rack locality.

Separate Clusters

With separate clusters, a Vertica cluster and a Hadoop cluster share no nodes. You should use a high-bandwidth network connection between the two clusters.

The following figure illustrates the configuration for separate clusters:
Network

The network is a key performance component of any well-configured cluster. When Vertica stores data to HDFS it writes and reads data across the network.

The layout shown in the figure calls for two networks, and there are benefits to adding a third:

- **Database Private Network:** Vertica uses a private network for command and control and moving data between nodes in support of its database functions. In some networks, the command and control and passing of data are split across two networks.

- **Database/Hadoop Shared Network:** Each Vertica node must be able to connect to each Hadoop data node and the Name Node. Hadoop best practices generally require a dedicated network for the Hadoop cluster. This is not a technical requirement, but a dedicated network improves Hadoop performance. Vertica and Hadoop should share the dedicated Hadoop network.
• Optional Client Network: Outside clients may access the clustered networks through a client network. This is not an absolute requirement, but the use of a third network that supports client connections to either Vertica or Hadoop can improve performance. If the configuration does not support a client network, then client connections should use the shared network.

Hadoop Configuration Parameters

For best performance, set the following parameters with the specified minimum values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS block size</td>
<td>512MB</td>
</tr>
<tr>
<td>Namenode Java Heap</td>
<td>1GB</td>
</tr>
<tr>
<td>Datanode Java Heap</td>
<td>2GB</td>
</tr>
</tbody>
</table>
Hadoop Interfaces

Vertica provides several ways to interact with data stored in HDFS, described below. Decisions about Cluster Layout can affect the decisions you make about which Hadoop interfaces to use.

Querying Data Stored in HDFS

Vertica can query data directly from HDFS without requiring you to copy data. To query data, you define an external table as for any other external data source. (This option does not make use of schema definitions from Hive.)

Vertica can query the data directly from the HDFS nodes, bypassing the WebHDFS service. To query data directly, Vertica needs access to HDFS configuration files.

See Reading Directly from HDFS and Reading Hadoop Columnar File Formats.

Querying Data Using the HCatalog Connector

The HCatalog Connector uses Hadoop services (Hive and HCatalog) to query data stored in HDFS. Using the HCatalog Connector, Vertica can use Hive's schema definitions. However, performance can be poor compared to defining your own external table and querying the data directly. The HCatalog Connector is also sensitive to changes in the Hadoop libraries on which it depends; upgrading your Hadoop cluster might affect your HCatalog connections.

See Using the HCatalog Connector.

Using ROS Data

Storing data in the Vertica native file format (ROS) delivers better query performance than reading externally-stored data. You can create storage locations on your HDFS cluster to hold ROS data. Even if your data is already stored in HDFS, you might choose to copy that data into an HDFS storage location for better performance. If your Vertica cluster also uses local file storage, you can use HDFS storage locations for lower-priority data.

See Using HDFS Storage Locations for information about creating and managing HDFS storage locations, and Reading Directly from HDFS for information about copying data into them.
Exporting Data

You might want to export data from Vertica, either to share it with other Hadoop-based applications or to move lower-priority data from ROS to less-expensive storage. You can export a table, or part of one, in Hadoop columnar format. After export you can still query the data using an external table, as for any other data.

See Exporting Data.
Using Kerberos with Hadoop

If your Hortonworks or Cloudera Hadoop cluster uses Kerberos authentication to restrict access to HDFS, you must configure Vertica to make authenticated connections. The details of this configuration vary, based on which methods you are using to access HDFS data.

Vertica does not support Kerberos for MapR.

This section covers the following topics:

- How Vertica uses Kerberos With Hadoop
- Configuring Kerberos

How Vertica uses Kerberos With Hadoop

Vertica authenticates with Hadoop in two ways that require different configurations:

- **User Authentication**—On behalf of the user, by passing along the user's existing Kerberos credentials. This method is also called user impersonation.

- **Vertica Authentication**—On behalf of system processes (such as the Tuple Mover), by using a special Kerberos credential stored in a keytab file.

  Note: Vertica and Hadoop must use the same Kerberos server or servers (KDCs).

User Authentication

To use Vertica with Kerberos and Hadoop, the client user first authenticates with one of the Kerberos servers (Key Distribution Center, or KDC) being used by the Hadoop cluster. A user might run kinit or sign in to Active Directory, for example.

A user who authenticates to a Kerberos server receives a Kerberos ticket. At the beginning of a client session, Vertica automatically retrieves this ticket. Vertica then uses this ticket to get a Hadoop token, which Hadoop uses to grant access. Vertica uses this token to access HDFS, such as when executing a query on behalf of the user. When the token expires, Vertica automatically renews it, also renewing the Kerberos ticket if necessary.

The user must have been granted permission to access the relevant files in HDFS. This permission is checked the first time Vertica reads HDFS data.
The following figure shows how the user, Vertica, Hadoop, and Kerberos interact in user authentication. The configuration in this example uses a single KDC. Vertica can also use multiple KDCs serving multiple Kerberos realms, if proper cross-realm trust has been set up between realms.

When reading an ORC or Parquet file stored in HDFS, and when using the HCatalog Connector under some conditions, Vertica uses the client identity as the preceding figure shows.

**Vertica Authentication**

Automatic processes, such as the Tuple Mover, do not log in the way users do. Instead, Vertica uses a special identity (principal) stored in a keytab file on every database node. (This approach is also used for Vertica clusters that use Kerberos but do not use Hadoop.) After you configure the keytab file, Vertica uses the principal residing there to automatically obtain and maintain a
Kerberos ticket, much as in the client scenario. In this case, the client does not interact with Kerberos.

The following figure shows the interactions required for Vertica authentication:

Each Vertica node uses its own principal; it is common to incorporate the name of the node into the principal name. You can either create one keytab per node, containing only that node's principal, or you can create a single keytab containing all the principals and distribute the file to all nodes. Either way, the node uses its principal to get a Kerberos ticket and then uses that ticket to get a Hadoop token. For simplicity, the preceding figure shows the full set of interactions for only one database node.

When creating HDFS storage locations Vertica uses the principal in the keytab file, not the principal of the user issuing the CREATE LOCATION statement. The HCatalog Connector sometimes uses the principal in the keytab file, depending on how Hive authenticates users.
See Also

For specific configuration instructions, see Configuring Kerberos.

Configuring Kerberos

Vertica can connect with Hadoop in several ways, and how you manage Kerberos authentication varies by connection type. If you use Kerberos, you must use it for both your HDFS and Vertica clusters.

Vertica can interact with more than one Kerberos realm. To configure multiple realms, see Multi-realm Support in Security and Authentication.

Prerequisite: Set Up Users and the Keytab File

If you have not already configured Kerberos authentication for Vertica, follow the instructions in Configure Vertica for Kerberos Authentication. Of particular importance for Hadoop integration:

1. Create one Kerberos principal per node.
2. Place the keytab files in the same location on each database node and set configuration parameter KerberosKeytabFile to that location.
3. Set KerberosServiceName to the name of the principal. (See Inform Vertica About the Kerberos Principal.)

Reads with the hdfs Scheme

Vertica can access files stored in HDFS using the hdfs URL scheme instead of using WebHDFS. Vertica authenticates using the current user's Kerberos principal, not the database's Kerberos principal. No additional Kerberos-specific configuration is required.
HCatalog Connector

You use the HCatalog Connector to query data in Hive. How you configure the HCatalog Connector depends on how Hive manages authorization.

- If Hive uses Sentry to manage authorization, and if Sentry uses ACL synchronization, then the HCatalog Connector must access Hive as the current user. Verify that the EnableHCatImpersonation configuration parameter is set to 1 (the default). ACL synchronization automatically provides authorized users with read access to the underlying HDFS files.

- If Hive uses Sentry without ACL synchronization, then the HCatalog Connector must access Hive data as the Vertica principal. (The user still authenticates and accesses metadata normally.) Set the EnableHCatImpersonation configuration parameter to 0. The Vertica principal must have read access through Sentry.

- If Hive uses Ranger to manage authorization, and the Vertica users have read access to the underlying HDFS files, then you can use user impersonation. Verify that the EnableHCatImpersonation configuration parameter is set to 1 (the default). You can, instead, disable user impersonation and give the Vertica principal read access to the HDFS files.

- If Hive uses either Sentry or Ranger, the HCatalog Connector must use HiveServer2 (the default). WebHCat does not support authorization services.

- If Hive does not use an authorization service, or if you are connecting to Hive using WebHCat instead of HiveServer2, then the HCatalog Connector accesses Hive as the current user. Verify that EnableHCatImpersonation is set to 1. All users must have read access to the underlying HDFS files.

In addition, in your Hadoop configuration files (core-site.xml in most distributions), make sure that you enable all Hadoop components to impersonate the Vertica user. The easiest way to do so is to set the proxyuser property using wildcards for all users on all hosts and in all groups. Consult your Hadoop documentation for instructions. Make sure you set this property before running hcatUtil (see Configuring Vertica for HCatalog).

HDFS Storage Location

You can create a database storage location in HDFS. An HDFS storage location provides improved performance compared to other HDFS interfaces (such as the HCatalog Connector).
By storing the data in Vertica, rather than creating an external table, you can reduce query response times.

To use a storage location in HDFS with Kerberos, take the following steps:

1. Create a Kerberos principal for each Vertica node as described under Prerequisites.

2. Give all node principals read and write permission to the HDFS directory you will use as a storage location.

If you plan to back up your HDFS storage locations, take the following additional steps:

1. Grant Hadoop superuser privileges to the new principals.

2. Configure backups, including setting the HadoopConfigDir configuration parameter, following the instructions in Configuring Hadoop and Vertica to Enable Backup of HDFS Storage.

3. Configure user impersonation to be able to restore from backups following the instructions in "Setting Kerberos Parameters" in Configuring Vertica to Restore HDFS Storage Locations.

Because the keytab file supplies the principal used to create the location, you must have it in place before creating the storage location. After you deploy keytab files to all database nodes, use the CREATE LOCATION statement to create the storage location as usual.

Verifying Kerberos Configuration

Use the KERBEROS_HDFS_CONFIG_CHECK metasfunction to verify that Vertica can use Kerberos to access HDFS. You can call it with no parameters to test all paths described in the Hadoop configuration files. Alternatively, you can specify hdfs, webhdfs, and WebHCat servers to test individually.

```sql
=> SELECT KERBEROS_HDFS_CONFIG_CHECK();
=> SELECT KERBEROS_HDFS_CONFIG_CHECK('node1.example.com:9433', 'node2.example.com:10443', 'node2.example.com:14443');
```

This function does not yet check access to HiveServer2.

Token Expiration

Vertica attempts to automatically refresh Hadoop tokens before they expire, but you can also set a minimum refresh frequency if you prefer. Use the HadoopFSTokenRefreshFrequency
configuration parameter to specify the frequency in seconds:

```sql
=> ALTER DATABASE exampledb SET HadoopFSTokenRefreshFrequency = '86400';
```

If the current age of the token is greater than the value specified in this parameter, Vertica refreshes the token before accessing data stored in HDFS.

See Also

- How Vertica uses Kerberos With Hadoop
- Troubleshooting Kerberos Authentication
Reading Directly from HDFS

When reading files from HDFS, you can use the hdfs scheme instead of the webhdfs scheme. Using the hdfs scheme can improve performance and stability by bypassing the WebHDFS service.

You can use the hdfs scheme with COPY and with CREATE EXTERNAL TABLE AS COPY. When using the hdfs scheme with COPY, you do not need to specify ON ANY NODE.

To support direct access, Vertica requires access to certain configuration files from your HDFS cluster. See Configuring the hdfs Scheme.

URL Format

You specify the location of a file in HDFS using a URL. In most cases, you use the hdfs:// URL prefix (three slashes), and then specify the file path. Vertica uses the fs.defaultFS Hadoop configuration parameter to find the data. The following example loads data stored in HDFS.

```sql
=> COPY t FROM 'hdfs:///opt/data/file1.dat';
```

You can specify a host and port explicitly using the following format: hdfs://host:port/. The specified host is the Name Node, not an individual data node. If you are using High Availability (HA) Name Nodes you should not use an explicit host because high availability is provided through nameservices instead.

Your HDFS cluster might use High Availability Name Nodes or define nameservices. If so, you should use the nameservice instead of the host and port, in the format hdfs://nameservice/. The nameservice you specify must be defined in hdfs-site.xml.

The following example shows how you can use a nameservice, hadoopNS, with the hdfs scheme.

```sql
=> CREATE EXTERNAL TABLE tt (a1 INT, a2 VARCHAR(20))
    AS COPY FROM 'hdfs://hadoopNS/data/file.csv';
```

If you are using Vertica to access data from more than one HDFS cluster, always use explicit nameservices or hosts in the URL. Using hdfs:// could produce unexpected results because Vertica uses the first value of fs.defaultFS that it finds. To access multiple HDFS clusters, you must use host and service names that are globally unique. See Configuring the hdfs Scheme for more information.
Kerberos Authentication

If the file you want to read resides on an HDFS cluster that uses Kerberos authentication, Vertica uses the current user's principal to authenticate. It does not use the database's principal.

You can use the KERBEROS_HDFS_CONFIG_CHECK metafunction to verify that Vertica is correctly configured for Kerberos access.

Configuring the hdfs Scheme

Vertica uses information from the Hadoop cluster configuration to support the hdfs scheme. Vertica nodes therefore must have access to certain Hadoop configuration files. To use a cluster with High Availability Name Node or to read from more than one Hadoop cluster, you must perform additional configuration.

For both co-located and separate clusters that use Kerberos authentication, configure Vertica for Kerberos as explained in Configure Vertica for Kerberos Authentication.

Using the hdfs scheme still requires access to the WebHDFS service. For some special cases, Vertica cannot use the hdfs scheme and falls back to webhdfs.

Accessing Hadoop Configuration Files

To support the hdfs scheme, your Vertica nodes need access to certain Hadoop configuration files:

- If Vertica is co-located on HDFS nodes, then those configuration files are already present.

- If Vertica is running on a separate cluster, you must copy the required files to all database nodes. A simple way to do so is to configure your Vertica nodes as Hadoop edge nodes. Client applications run on edge nodes; from Hadoop's perspective, Vertica is a client application. You can use Ambari or Cloudera Manager to configure edge nodes. For more information, see the documentation from your Hadoop vendor.
Verify that the value of the HadoopConfDir configuration parameter (see Apache Hadoop Parameters) includes a directory containing the files listed in the following table. If you do not set a value, Vertica looks for the files in /etc/hadoop/conf. You can use the VERIFY_HADOOP_CONF_DIR meta-function to verify that Vertica can find configuration files.

Vertica uses the following configuration files and properties. If a property is not defined, Vertica uses the defaults shown in the table. Your Hadoop configuration files must specify all properties that have no defaults.

<table>
<thead>
<tr>
<th>File</th>
<th>Properties</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>core-site.xml</td>
<td>fs.defaultFS</td>
<td>none</td>
</tr>
<tr>
<td>hdfs-site.xml</td>
<td>dfs.client.failover.max.attempts</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>dfs.client.failover.connection.retries.on.timeouts</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ipc.client.connect.timeout</td>
<td>30 seconds</td>
</tr>
<tr>
<td></td>
<td>ipc.client.connect.retry.interval</td>
<td>10 seconds</td>
</tr>
<tr>
<td>(For HA NN:)</td>
<td>dfs.nameservices</td>
<td>none</td>
</tr>
</tbody>
</table>

Using a Cluster with High Availability Name Nodes

If your Hadoop cluster uses High Availability (HA) Name Nodes, verify that the dfs.nameservices parameter and the individual name nodes are defined in hdfs-site.xml.

Using More Than One Hadoop Cluster

In some cases, a Vertica cluster requires access to more than one HDFS cluster. For example, your business might use separate HDFS clusters for separate regions, or you might need data from both test and deployment clusters.

To support multiple clusters, perform the following steps:

1. Copy the configuration files from all HDFS clusters to your database nodes. You can place the copied files in any location readable by Vertica. However, as a best practice, you should place them all in the same directory tree, with one subdirectory per HDFS cluster. The locations must be the same on all database nodes.
2. Set the HadoopConfDir configuration parameter. The value is a colon-separated path containing the directories for all of your HDFS clusters.

3. Use an explicit name node or name service in the URL when creating an external table or copying data. Do not use hdfs:/// because it could be ambiguous. For more information about URLs, see URL Format.

Vertica connects directly to a name node or name service; it does not otherwise distinguish among HDFS clusters. Therefore, names of HDFS name nodes and name services must be globally unique.

Updating Configuration Files

If you update the configuration files after starting Vertica, use the following statement to refresh them:

```sql
=> SELECT CLEAR_HDFS_CACHES();
```

The CLEAR_HDFS_CACHES function also flushes information about which Name Node is active in a High Availability (HA) Hadoop cluster. Therefore, the first hdfs request after calling this function is slow, because the initial connection to the Name Node can take more than 15 seconds.

Troubleshooting Reads from the hdfs Scheme

You might encounter the following issues when using the hdfs URL scheme to access data in HDFS.

WebHDFS Error When Using hdfs URLs

When creating an external table or loading data and using the hdfs scheme, you might see errors from WebHDFS failures. Such errors indicate that Vertica was not able to use the hdfs scheme and fell back to webhdfs, but that the WebHDFS configuration is incorrect. Verify that the HDFS configuration files in HadoopConfDir have the correct WebHDFS configuration for your Hadoop cluster. See Configuring the hdfs Scheme for information about use of these files. See your Hadoop documentation for information about WebHDFS configuration.
Queries Take a Long Time to Run When Using HA

The High Availability Name Node feature in HDFS allows a name node to fail over to a standby name node. The dfs.client.failover.max.attempts configuration parameter (in hdfs-site.xml) specifies how many attempts to make when failing over. Vertica uses a default value of 4 if this parameter is not set. After reaching the maximum number of failover attempts, Vertica concludes that the HDFS cluster is unavailable and aborts the operation. A second parameter, ipc.client.connect.retry.interval, specifies the time to wait between attempts, with typical values being 10 to 20 seconds.

Cloudera and Hortonworks both provide tools to automatically generate configuration files. These tools can set the maximum number of failover attempts to a much higher number (50 or 100). If the HDFS cluster is unavailable (all name nodes are unreachable), Vertica can appear to hang for an extended period (minutes to hours) while trying to connect.

Failover attempts are logged in the QUERYEVENTS system table. The following example shows how to query this table to find these events:

```sql
=> SELECT event_category, event_type, event_description, operator_name,
     event_details, count(event_type) AS count
     FROM query_events
     WHERE event_type ilike 'LibHDFS++ FAILOVER RETRY'
     GROUP BY event_category, event_type, event_description, operator_name, event_details;
```

<table>
<thead>
<tr>
<th>event_category</th>
<th>EXECUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_type</td>
<td>LibHDFS++ FAILOVER RETRY</td>
</tr>
<tr>
<td>event_description</td>
<td>LibHDFS++ Namenode failover and retry.</td>
</tr>
<tr>
<td>operator_name</td>
<td>LibHDFS++ FileSystem</td>
</tr>
<tr>
<td>event_details</td>
<td>Libhdfs++ request failed on ns</td>
</tr>
<tr>
<td>count</td>
<td>4</td>
</tr>
</tbody>
</table>

You can either wait for Vertica to complete or abort the connection, or set the dfs.client.failover.max.attempts parameter to a lower value.
Reading Hadoop Columnar File Formats

For data in certain Hadoop columnar formats, you can create external tables or copy data into Vertica directly, without going through Hive to get the metadata. Currently, Vertica supports direct reading for the ORC (Optimized Row Columnar) and Parquet formats. Vertica can read ORC and Parquet data from a local file system or from HDFS.

Vertica can also export data in the Parquet format. See Exporting Data.

Requirements

ORC or Parquet files must not use complex data types. Vertica supports all simple data types supported in Hive version 0.11 or later.

Files compressed by Hive or Impala require Zlib (GZIP) or Snappy compression. Vertica does not support LZO compression for these formats.

Privileges

Files stored in HDFS are governed by HDFS privileges.

For files stored on the local disk, Vertica requires that users be granted access. All users who have administrative privileges have access. For other users, you must create a storage location and grant access to it. See CREATE EXTERNAL TABLE AS COPY. Only users who have access to both the Vertica user storage location and the HDFS directory can read from the table.

For files stored in an S3 bucket, users must have access through the AWS IAM role. For more information about IAM roles, see Amazon's Web Services documentation.

Creating External Tables

In the CREATE EXTERNAL TABLE AS COPY statement, specify a format of ORC or PARQUET as follows:

```sql
=> CREATE EXTERNAL TABLE tableName (columns)
    AS COPY FROM path ORC([hive_partition_cols='partitions']);
=> CREATE EXTERNAL TABLE tableName (columns)
```
AS COPY FROM path PARQUET[(hive_partition_cols='partitions')];

Set the value of path based on where the file is located. If the file resides:

- On the local file system of the node where you issue the command—Use a local file path. Escape special characters in file paths with backslashes.

- In HDFS—Use the hdfs scheme and then append the file path. Escape special characters in HDFS paths using URL encoding. HDFS URLs usually begin with hdfs:///. See Reading Directly from HDFS for more information about URL format.

- In an S3 bucket—Use a URL of the form "S3://bucket/path". See Loading from an S3 Bucket in the Administrator’s Guide.

When defining an external table, you must define all of the data columns in the file. You may omit partition columns. Unlike with some other data sources, you cannot select only the data columns of interest. If you omit data columns, the ORC or Parquet reader aborts with an error.

If the data is partitioned you must alter the path value and specify the hive_partition_cols argument for the ORC or PARQUET parameter. You must also list partitioned columns last in columns. See Using Partition Columns.

If path is a path on the local file system on a Vertica node, specify the node using ON NODE in the COPY statement. Do not use COPY LOCAL. If path is in HDFS or S3, COPY defaults to ON ANY NODE so you do not need to specify it.

### Loading Data

In the COPY statement, specify a format of ORC or PARQUET:

```sql
=> COPY tableName FROM path ORC[(hive_partition_cols='partitions')];
=> COPY tableName FROM path PARQUET[(hive_partition_cols='partitions')];
```

As with external tables, path may be a local, hdfs:, or S3: path. For information about the hive_partition_cols parameter, see Using Partition Columns.

Be aware that if you load from multiple files in the same COPY statement, and any of them is aborted, the entire load aborts. This behavior differs from that for delimited files, where the COPY statement loads what it can and ignores the rest.
Supported Data Types

Vertica can natively read columns of all data types supported in Hive version 0.11 and later except for complex types. If the data contains complex types such as maps, the COPY or CREATE EXTERNAL TABLE AS COPY statement aborts with an error message. Vertica does not attempt to read only some columns; either the entire file is read or the operation fails. For a complete list of supported types, see HIVE Data Types.

The data types you specify for COPY or CREATE EXTERNAL TABLE AS COPY must exactly match the types in the ORC or Parquet data. Vertica treats DECIMAL and FLOAT as the same type, but they are different in the ORC and Parquet formats and you must specify the correct one.

Versions of Hive before 1.2.1 wrote TIMESTAMP values in UTC. Newer versions of Hive can write timestamps in local time or UTC, but this information is not in the metadata. By default Vertica assumes the values are in UTC and converts them to the local time zone at read time. When reading Parquet files with local time zones instead of UTC, set the UseLocalTzForParquetTimestampConversion configuration parameter to 0 to disable the conversion done by Vertica. (See General Parameters.)

Examples

The following example shows how you can read from all ORC files in a local directory. This example uses all supported data types.

```sql
=> CREATE EXTERNAL TABLE t (a1 TINYINT, a2 SMALLINT, a3 INT, a4 BIGINT, a5 FLOAT,
a6 DOUBLE PRECISION, a7 BOOLEAN, a8 DATE, a9 TIMESTAMP,
a10 VARCHAR(20), a11 VARCHAR(20), a12 CHAR(20), a13 BINARY(20),
a14 DECIMAL(10,5))
AS COPY FROM '/data/orc_test_*\.orc' ORC;
```

The following example shows how to use a name service with the hdfs scheme. This example assumes that the name service, hadoopNS, is defined in the Hadoop configuration files that were copied to the Vertica cluster.

```sql
=> CREATE EXTERNAL TABLE tt (a1 INT, a2 VARCHAR(20))
AS COPY FROM 'hdfs://hadoopNS/data/file.parquet' PARQUET;
```

The following example shows how to load multiple ORC files from one S3 bucket.

```sql
=> SELECT SET_CONFIG_PARAMETER('AWSRegion','us-west-1');
```
Using Partition Columns

An ORC or Parquet file contains data columns. To these files you can add partition columns at write time. The data files do not store values for partition columns; instead, when writing the files you divide them into groups (partitions) based on column values. You can use partitioning to improve the performance of queries that restrict results by the partitioned column.

For example, if you have a table with a date column, and you know you will be writing queries restricted to particular dates, you can partition by date. Thus, Vertica can skip reading some files entirely when executing your date-restricted queries. This behavior is called partition pruning.

You can create partitions regardless of where you store the files—in HDFS, in an S3 bucket, on a local file system, or in a shared file system such as NFS.

You can use Hive or the Vertica Parquet Writer to create partitions, or you can create them manually. See Partitioning Hive Tables for information about tuning partitions.

Partition Structure

By default, both Hive and Vertica write Hadoop columnar format files that contain the data for all table columns without partitioning. The column data is laid out in stripes, or groups of row data. When Vertica loads this data it reads all of the stripes.

If you partition the data, however, you can avoid writing some of that data into the files and thus reduce the amount to be read. Instead of storing a column's data in the files, you create a directory structure that partitions the data based on the value in a column.

For example, if the data includes a date column, you can write each date as a separate partition. Each partition is a directory with a name of the form "column=value". If you have a date column named "created" that is partitioned by day, you would have the following directory structure:

```
path/created=2016-11-01/*
path/created=2016-11-02/*
path/created=2016-11-03/*
path/...
```

As this example shows, the files in each subdirectory contain all columns except the "created" column.
You can partition by more than one column, creating a layered structure. For example, adding another partitioned column, "region", to the preceding example would produce the following directory structure:

```
path/created=2016-11-01/region=northeast/*
path/created=2016-11-01/region=central/*
path/created=2016-11-01/region=southeast/*
path/created=2016-11-01/...
path/created=2016-11-02/region=northeast/*
path/created=2016-11-02/region=central/*
path/created=2016-11-02/region=southeast/*
path/created=2016-11-02/...
path/created=2016-11-03/...
path/...
```

With this change, the data files contain all columns except "created" and "region".

**Note:** The files must contain at least one real (not partitioned) column. You cannot partition by every column in a table.

You can create partitions for columns of any simple data type. As a best practice, however, you should avoid partitioning columns with BOOLEAN, FLOAT, and NUMERIC types. Vertica does not prune partitions of these types, so they do not provide a performance advantage.

Under some circumstances Hive writes a partition with a value of `___HIVE_DEFAULT_PARTITION__`. Vertica treats these values as NULL.

### COPY Syntax

When creating an external table from partitioned data, you must do all of the following:

- In the column definition in the table, list the partition columns last and in order.
- In the path, use wildcards to include all of the levels of directories and files.
- In the ORC or PARQUET statement, specify the partition columns, in order, in the `hive_partition_cols` parameter. (The argument name is the same even if you didn't use Hive to do the partitioning; it refers to Hive-style partitions.)

The following example creates an external table using the partitioned data shown previously. The table includes four columns. Two columns, "id" and "name", are in the data files. The other two, "created" and "region", are partitioned.

```
=> CREATE EXTERNAL TABLE t (id int, name varchar(50), created date, region varchar(50))
   AS COPY FROM 'hdfs:///path/*/*/*'
   ORC(hive_partition_cols='created,region');
```
The path includes one wildcard (*) for each level of directory partitioning and then one more for the files. The number of wildcards must always be one more than the number of partitioned columns.

You do not need to include all of the partitioned columns in hive_partition_cols if those columns are not relevant for your queries. However, the partition columns must be the last columns in the table definition. For example, you can define the following table for the partitioned data shown previously:

```sql
=> CREATE EXTERNAL TABLE t2 (id int, name varchar(50), created date, region varchar(50))
   AS COPY 'hdfs:///path/*//*' ORC(hive_partition_cols='region');
```

Values in the "created" column are all null because no data appears in the files for that column and hive_partition_cols does not include it.

However, the following example produces an error.

```sql
=> CREATE EXTERNAL TABLE t3 (id int, name varchar(50), created date, region varchar(50))
   AS COPY 'hdfs:///path/*//*' ORC(hive_partition_cols='created');
```

In this example, the table definition includes the "region" column after the "created" column, and "region" is not included in hive_partition_cols. Because this column is not listed as a partition column, Vertica interprets it as a data column and produces an error because the column is not present.

If Vertica cannot convert a partition value to the declared type for that column, it sets the value to NULL. The following example incorrectly declares region to be an integer rather than a varchar.

```sql
=> CREATE EXTERNAL TABLE t4 (id int, name varchar(50), created date, region int)
   AS COPY 'hdfs:///path/*//*' ORC(hive_partition_cols='region');
```

Vertica cannot coerce a directory named "region=northeast" into an integer value, so it sets that column value to NULL for all rows it reads from this directory. If you declare the column with IS NOT NULL, Vertica rejects the row. If the number of rows exceeds REJECTMAX, Vertica reports an error.

Note: If you change how files are partitioned on disk, you must re-create your external tables.

Queries

When executing queries with predicates, Vertica skips subdirectories that do not satisfy the predicate. This process is called partition pruning and it can significantly improve query
performance. See Improving Query Performance for Data Stored in HDFS for more information about partition pruning and other techniques for optimizing queries.

The following example reads only the partitions for the specified region, for all dates. Although the data is also partitioned by date, the query does not restrict the date.

```sql
=> SELECT * FROM t WHERE region='northeast';
```

To verify that Vertica is pruning partitions, look in the explain plan for a message similar to the following:

files with unmatched Hive partition have been pruned

---

**Improving Query Performance for Data Stored in HDFS**

When working with external tables in Hadoop columnar formats, Vertica tries to improve performance in the following ways:

- By pushing query execution closer to the data so less has to be read and transmitted. Vertica uses the following specific techniques: predicate pushdown, column selection, and partition pruning.
- By taking advantage of data locality in the query plan.
- By analyzing the row count to get the best join orders in the query plan.

The following figure illustrates optimizations that can reduce the amount of data to be read:
Tuning ORC Stripes and Parquet Rowgroups

Vertica can read ORC and Parquet files generated by any Hive version. However, newer Hive versions store more metadata in these files. This metadata is used by both Hive and Vertica to prune values and to read only the required data. Use the latest Hive version to store data in these formats. ORC and Parquet are fully forward- and backward-compatible. To get the best performance, use Hive 0.14 or later.

The ORC format splits a table into groups of rows called stripes and stores column-level metadata in each stripe. The Parquet format splits a table into groups of rows called rowgroups and stores column-level metadata in each rowgroup. Each stripe/rowgroup's metadata is used during predicate evaluation to determine whether the values from this stripe need to be read. Large stripes usually yield better performance, so set the stripe size to at least 256M.

Hive writes ORC stripes and Parquet rowgroups to HDFS, which stores data in HDFS blocks distributed among multiple physical data nodes. Accessing an HDFS block requires opening a separate connection to the corresponding data node. It is advantageous to ensure that an ORC stripe or Parquet rowgroup does not span more than one HDFS block. To do so, set the HDFS block size to be larger than the stripe/rowgroup size. Setting HDFS block size to 512M is usually sufficient.

Hive provides three compression options: None, Snappy, and Zlib. Use Snappy or Zlib compression to reduce storage and I/O consumption. Usually, Snappy is less CPU-intensive but can yield lower compression ratios compared to Zlib.

Storing data in sorted order can improve data access and predicate evaluation performance. Sort table columns based on the likelihood of their occurrence in query predicates; columns that most frequently occur in comparison or range predicates should be sorted first.
Partitioning tables is a very useful technique for data organization. Similarly to sorting tables by columns, partitioning can improve data access and predicate evaluation performance. Vertica supports Hive-style partitions and partition pruning.

The following Hive statement creates an ORC table with stripe size 256M and Zlib compression:

```sql
hive> CREATE TABLE customer_visits (
    customer_id bigint,
    visit_num int,
    page_view_dt date)
STORRED AS ORC tblproperties(
    "orc.compress"="ZLIB",
    "orc.stripe.size"="268435456");
```

The following statement creates a Parquet table with stripe size 256M and Zlib compression:

```sql
hive> CREATE TABLE customer_visits (
    customer_id bigint,
    visit_num int,
    page_view_dt date)
STORRED AS PARQUET tblproperties(
    "parquet.compression"="ZLIB",
    "parquet.stripe.size"="268435456");
```

## Predicate Pushdown and Column Selection

*Predicate pushdown* moves parts of the query execution closer to the data, reducing the amount of data that must be read from disk or across the network. ORC files have three levels of indexing: file statistics, stripe statistics, and row group indexes. Predicates are applied only to the first two levels. Parquet files have two levels of statistics: rowgroup statistics and page statistics. Predicates are only applied to the first level.

Predicate pushdown is automatically applied for files written with Hive version 0.14 and later. ORC files written with earlier versions of Hive might not contain the required statistics. When executing a query against a file that lacks these statistics, Vertica logs an `EXTERNAL_PREDICATE_PUSHDOWN_NOT_SUPPORTED` event in the `QUERY_EVENTS` system table. If you are seeing performance problems with your queries, check this table for these events.

Another query performance optimization technique used by Vertica is *column selection*. Vertica reads from ORC or Parquet files only the columns specified in the query statement. For example, the following statement reads only the `customer_id` and `visit_num` columns from the corresponding ORC files:

```sql
=> CREATE EXTERNAL TABLE customer_visits (
    customer_id bigint,
    visit_num int,
    page_view_dt date)
AS COPY FROM '...' ORC;

=> SELECT customer_id from customer_visits
```
WHERE visit_num > 10;

Data Locality

In a cluster where Vertica nodes are co-located on HDFS nodes, the query can use data locality to improve performance. For Vertica to do so, both the following conditions must exist:

- The data is on an HDFS node where a database node is also present.
- The query is not restricted to specific nodes using ON NODE.

When both these conditions exist, the query planner uses the co-located database node to read that data locally, instead of making a network call.

You can see how much data is being read locally by inspecting the query plan. The label for LoadStep(s) in the plan contains a statement of the form: "X% of ORC/Parquet data matched with co-located Vertica nodes". To increase the volume of local reads, consider adding more database nodes. HDFS data, by its nature, can't be moved to specific nodes, but if you run more database nodes you increase the likelihood that a database node is local to one of the copies of the data.

Creating Sorted Files in Hive

Unlike Vertica, Hive does not store table columns in separate files and does not create multiple projections per table with different sort orders. For efficient data access and predicate pushdown, sort Hive table columns based on the likelihood of their occurrence in query predicates. Columns that most frequently occur in comparison or range predicates should be sorted first.

Data can be inserted into Hive tables in a sorted order by using the ORDER BY or SORT BY keywords. For example, to insert data into the ORC table "customer_visit" from another table "visits" with the same columns, use these keywords with the INSERT INTO command:

```
hive> INSERT INTO TABLE customer_visits
    SELECT * from visits
    ORDER BY page_view_dt;
```

```
hive> INSERT INTO TABLE customer_visits
    SELECT * from visits
    SORT BY page_view_dt;
```
The difference between the two keywords is that ORDER BY guarantees global ordering on the entire table by using a single MapReduce reducer to populate the table. SORT BY uses multiple reducers, which can cause ORC or Parquet files to be sorted by the specified column(s) but not be globally sorted. Using the latter keyword can increase the time taken to load the file.

You can combine clustering and sorting to sort a table globally. The following table definition adds a hint that data is inserted into this table bucketed by customer_id and sorted by page_view_dt:

```
hive> CREATE TABLE customer_visits_bucketed (  
    customer_id bigint,  
    visit_num int,  
    page_view_dt date)  
CLUSTERED BY (page_view_dt)  
SORTED BY (page_view_dt)INTO 10 BUCKETS  
STORED AS ORC;
```

When inserting data into the table, you must explicitly specify the clustering and sort columns, as in the following example:

```
hive> INSERT INTO TABLE customer_visits_bucketed  
SELECT * from visits  
DISTRIBUTE BY page_view_dt  
SORT BY page_view_dt;
```

The following statement is equivalent:

```
hive> INSERT INTO TABLE customer_visits_bucketed  
SELECT * from visits  
CLUSTER BY page_view_dt;
```

Both of the above commands insert data into the customer_visits_bucketed table, globally sorted on the page_view_dt column.

**Partitioning Hive Tables**

Table partitioning in Hive is an effective technique for data separation and organization, as well as for reducing storage requirements. To partition a table in Hive, include it in the PARTITIONED BY clause:

```
hive> CREATE TABLE customer_visits (  
    customer_id bigint,  
    visit_num int)  
PARTITIONED BY (page_view_dt date)  
STORED AS ORC;
```

Hive does not materialize partition column(s). Instead, it creates subdirectories of the following form:
When the table is queried, Hive parses the subdirectories' names to materialize the values in the partition columns. The value materialization in Hive is a plain conversion from a string to the appropriate data type.

Inserting data into a partitioned table requires specifying the value(s) of the partition column(s). The following example creates two partition subdirectories, "customer_visits/page_view_dt=2016-02-01" and "customer_visits/page_view_dt=2016-02-02":

```sql
hive> INSERT INTO TABLE customer_visits
    PARTITION (page_view_dt='2016-02-01')
    SELECT customer_id, visit_num from visits
    WHERE page_view_dt='2016-02-01'
    ORDER BY page_view_dt;

hive> INSERT INTO TABLE customer_visits
    PARTITION (page_view_dt='2016-02-02')
    SELECT customer_id, visit_num from visits
    WHERE page_view_dt='2016-02-02'
    ORDER BY page_view_dt;
```

Each directory contains ORC files with two columns, customer_id and visit_num.

**Accessing Partitioned Data from Vertica**

Vertica recognizes and supports Hive-style partitions. You can read partition values and data using the HCatalog Connector or the COPY statement.

If you use the HCatalog Connector, you must create an HCatalog schema in Vertica that mirrors a schema in Hive:

```sql
=> CREATE EXTERNAL TABLE customer_visits (customer_id int, visit_num int, 
    page_view_dtm date)
    AS COPY FROM 'hdfs://host:port/path/customer_visits/*/*' ORC
    (hive_partition_cols='page_view_dtm');
```

The following statement reads all ORC files stored in all sub-directories including the partition values:

```sql
=> SELECT customer_id, visit_num, page_view FROM customer_visits;
```

When executing queries with predicates on partition columns, Vertica uses the subdirectory names to skip files that do not satisfy the predicate. This process is called *partition pruning*.

You can also define a separate external table for each subdirectory, as in the following example:
Example: A Partitioned, Sorted ORC Table

Suppose you have data stored in CSV files containing three columns: customer_id, visit_num, page_view_dtm:

1,123,2016-01-01
33,1,2016-02-01
2,57,2016-01-03
...

The goal is to create the following Hive table:

```sql
hive> CREATE TABLE customer_visits (customer_id bigint, visit_num int) PARTITIONED BY (page_view_dtm date) STORED AS ORC;
```

To achieve this, perform the following steps:

1. Copy or move the CSV files to HDFS.
2. Define a textfile Hive table and copy the CSV files into it:

```sql
hive> CREATE TABLE visits (customer_id bigint, visit_num int, page_view_dtm date) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',', STORED AS TEXTFILE;

hive> LOAD DATA INPATH path_to_csv_files INTO TABLE visits;
```

3. For each unique value in page_view_dtm, insert the data into the target table while materializing page_view_dtm as page_view_dtm:

```sql
hive> INSERT INTO TABLE customer_visits PARTITION (page_view_dtm='2016-01-01') SELECT customer_id, visit_num FROM visits WHERE page_view_dtm='2016-01-01' ORDER BY page_view_dtm;
```
This operation inserts data from visits.customer_id into customer_visits.customer_id, and from visits.visit_num into customer_visits.visit_num. These two columns are stored in generated ORC files. Simultaneously, values from visits.page_view_dt are used to create partitions for the partition column customer_visits.page_view_dt, which is not stored in the ORC files.

Data Modification in Hive

Hive is well-suited for reading large amounts of write-once data. Its optimal usage is loading data in bulk into tables and never modifying the data. In particular, for data stored in the ORC and Parquet formats, this usage pattern produces large, globally (or nearly globally) sorted files.

Periodic addition of data to tables (known as “trickle load”) is likely to produce many small files. The disadvantage of this is that Vertica has to access many more files during query planning and execution. These extra access can result in longer query-processing time. The major performance degradation comes from the increase in the number of file seeks on HDFS.

Hive can also modify underlying ORC or Parquet files without user involvement. If enough records in a Hive table are modified or deleted, for example, Hive deletes existing files and replaces them with newly-created ones. Hive can also be configured to automatically merge many small files into a few larger files.

When new tables are created, or existing tables are modified in Hive, you must manually synchronize Vertica to keep it up to date. The following statement synchronizes the Vertica schema "hcat" after a change in Hive:

```sql
=> SELECT sync_with_hcatalog_schema('hcat_local', 'hcat');
```

Schema Evolution in Hive

Hive supports two kinds of schema evolution:

1. New columns can be added to existing tables in Hive. Vertica automatically handles this kind of schema evolution. The old records display NULLs for the newer columns.

2. The type of a column for a table can be modified in Hive. Vertica does not support this kind of schema evolution.

The following example demonstrates schema evolution through new columns. In this example, hcat.parquet.txt is a file with the following values:
hive> create table hcat.parquet_tmp (a int, b float, c double, d int, e varchar(4))
    row format delimited fields terminated by '|' lines terminated by '\n';

hive> load data local inpath 'hcat.parquet.txt' overwrite into table hcat.parquet_tmp;

hive> create table hcat.parquet_evolve (a int) partitioned by (f int) stored as parquet;
hive> insert into table hcat.parquet_evolve partition (f=1) select a from hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (b float);
hive> insert into table hcat.parquet_evolve partition (f=2) select a, b from hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (c double);
hive> insert into table hcat.parquet_evolve partition (f=3) select a, b, c from hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (d int);
hive> insert into table hcat.parquet_evolve partition (f=4) select a, b, c, d from hcat.parquet_tmp;
hive> alter table hcat.parquet_evolve add columns (e varchar(4));
hive> insert into table hcat.parquet_evolve partition (f=5) select a, b, c, d, e from hcat.parquet_tmp;
hive> insert into table hcat.parquet_evolve partition (f=6) select a, b, c, d, e from hcat.parquet_tmp;

=> SELECT * from hcat_local.parquet_evolve;

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td>0.65</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td>0.65</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td>0.65</td>
<td>6</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>-1</td>
<td>0.64999976158142</td>
<td>0.65</td>
<td>6</td>
<td>b</td>
<td>6</td>
</tr>
</tbody>
</table>

(6 rows)

Troubleshooting Reads from Native File Formats

You might encounter the following issues when reading ORC or Parquet files.
Reads from Parquet Files Report Unexpected Data-Type Mismatches

If a Parquet file contains a column of type STRING but the column in Vertica is of a different type, such as INTEGER, you might see an unclear error message. In this case Vertica reports the column in the Parquet file as BYTE_ARRAY, as shown in the following example:

```
ERROR @: Datatype mismatch: column 2 in the parquet_cpp source [/tmp/nation.0.parquet] has type BYTE_ARRAY, expected int
```

This behavior is specific to Parquet files; with an ORC file the type is correctly reported as STRING. The problem occurs because Parquet does not natively support the STRING type and uses BYTE_ARRAY for strings instead. Because the Parquet file reports its type as BYTE_ARRAY, Vertica has no way to determine if the type is actually a BYTE_ARRAY or a STRING.

Time Zones in Timestamp Values Are Not Correct

Reading time stamps from an ORC or Parquet file in Vertica might result in different values, based on the local time zone. This issue occurs because the ORC and Parquet formats do not support the SQL TIMESTAMP data type. If you define the column in your table with the TIMESTAMP data type, Vertica interprets time stamps read from ORC or Parquet files as values in the local time zone. This same behavior occurs in Hive. When this situation occurs, Vertica produces a warning at query time, such as the following:

```
WARNING @: SQL TIMESTAMPTZ is more appropriate for ORC TIMESTAMP
because values are stored in UTC
```

When creating the table in Vertica, you can avoid this issue by using the TIMESTAMPTZ data type instead of TIMESTAMP.

Some Date and Timestamp Values Are Wrong by Several Days

When Hive writes ORC or Parquet files, it converts dates before 1583 from the Gregorian calendar to the Julian calendar. Vertica does not perform this conversion. If your file contains dates before this time, values in Hive and the corresponding values in Vertica can differ by up to ten days. This difference applies to both DATE and TIMESTAMP values.
Error 7087: Wrong Number of Columns

When loading data, you might see an error stating that you have the wrong number of columns:

```sql
=> CREATE TABLE nation (nationkey bigint, name varchar(500),
    regionkey bigint, comment varchar(500));
CREATE TABLE

=> COPY nation from :orc_dir ORC;
ERROR 7087: Attempt to load 4 columns from an orc source
[/tmp/orc_glob/test.orc] that has 9 columns
```

When you load data from Hadoop native file formats, your table must consume all of the data in the file, or this error results. To avoid this problem, add the missing columns to your table definition.

Error 7226: Cannot Find Partition Column

When querying data, you might see an error message stating that a partition column is missing:

```sql
ERROR 7226: Cannot find partition column [region] in parquet source
[/data/table_int/int_original/000000_0]
```

This error can occur if you partition your ORC or Parquet data (see Using Partition Columns). If you create an external table and then change the partition structure, for example by renaming a column, you must then re-create the external table. If you see this error, update your table to match the partitioning on disk.

Error 6766: Is a Directory

When querying data that is stored on the local disk, you might see an error message stating that an input file is a directory:

```sql
ERROR 6766: Error reading from orc parser input stream
[/tmp/orc_glob/more_nations]: Is a directory
```

This error occurs if the glob in the table's COPY FROM clause matches an empty directory. This error occurs only for files in the Linux file system; empty directories in HDFS are ignored.

To correct the error, make the glob more specific. Instead of *, for example, use *.orc.
Exporting Data

You might want to export data from Vertica, either to share it with other Hadoop-based applications or to move lower-priority data from ROS to less-expensive storage. You can use the EXPORT TO PARQUET statement to export a table (or part of one) as Parquet data.

You can export data to HDFS or to the local file system. You can export ROS data or data that is readable through external tables. After exporting ROS data, you can drop affected ROS partitions to reclaim storage space. If you need to access the data in Vertica again, you can create external tables from the exported data.

To export data, use EXPORT TO PARQUET in combination with a SELECT statement, as in the following example:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data')
   AS SELECT * FROM public.sales;
Rows Exported
---------------
  14336
(1 row)
```

The directory argument specifies where to write the files and is required. The directory must not already exist.

You can use EXPORT TO PARQUET to write queries across multiple tables in Vertica and export the results. With this approach you can take advantage of powerful, fast query execution in Vertica while making the results available to other Hadoop clients:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_by_region')
   AS SELECT sale.price, sale.date, store.region
        FROM public.sales sale
        JOIN public.vendor store ON sale.distribID = store.ID;
Rows Exported
---------------
   23301
(1 row)
```

EXPORT TO PARQUET takes optional parameters to specify compression format and row-group size (in MB), as in the following example:

```
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data',
   compression = 'uncompressed', rowGroupSize = '32')
   AS SELECT * FROM public.sales;
Rows Exported
---------------
   14336
(1 row)
```

The default compression type is Snappy.
The row-group size affects memory consumption during export. An export thread consumes at least 64MB of RAM if the value is 64. The default value of 64 is a compromise between writing larger row groups and allowing enough free memory for other Vertica operations. If you perform exports when the database is not otherwise under heavy load, you can improve read performance later by increasing row-group size on export.

When exporting, you can use the optional OVER clause to specify how to partition and/or sort data. Partitioning reduces the sizes of the output data files and can improve performance when Vertica queries external tables containing this data. (See Using Partition Columns.) If you do not specify how to partition the data, Vertica optimizes the export for maximum parallelism.

To specify partition columns, use PARTITION BY in the OVER clause as in the following example:

```sql
=> EXPORT TO PARQUET(directory = 'hdfs:///data/export')
    OVER(PARTITION BY date) AS SELECT date, price FROM public.sales;
Rows Exported
          ----------
           28337
(1 row)
```

You can sort values within each partition for a further performance improvement. Sort table columns based on the likelihood of their occurrence in query predicates; columns that most frequently occur in comparison or range predicates should be sorted first. You can sort values within each partition using ORDER BY in the OVER clause:

```sql
=> EXPORT TO PARQUET(directory = 'hdfs:///data/export')
    OVER(PARTITION BY date ORDER BY price) AS SELECT date, price FROM public.sales;
Rows Exported
          ----------
           28337
(1 row)
```

You can use ORDER BY even without partitioning. Storing data in sorted order can improve data access and predicate evaluation performance.

Targets in the OVER clause must be column references; they cannot be expressions. For more information about OVER, see SQL Analytics.

If you are exporting data to a local file system, you might want to force a single node to write all of the files. To do so, use an empty OVER clause.

You cannot export columns with the TIME, TIMEZ, and INTERVAL data types. If your table includes columns of these types, exclude them by explicitly selecting the columns you can export:

```sql
=> EXPORT TO PARQUET(directory='hdfs:///data/sales_data')
    AS SELECT date, transactionID, price FROM public.sales;
Rows Exported
          ----------
           14336
```
You can only perform one export per output directory. If you perform more than one concurrent export to the same directory, only one will succeed.

**Data Types**

EXPORT TO PARQUET converts Vertica data types to Hive data types as shown in the following table.

<table>
<thead>
<tr>
<th>Vertica Data Type</th>
<th>Hive Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER, BIGINT</td>
<td>BIGINT</td>
</tr>
<tr>
<td>FLOAT, DECIMAL, SMALLINT, TINYINT, CHAR, BOOLEAN</td>
<td>Corresponding Hive type</td>
</tr>
<tr>
<td>VARCHAR, LONG VARCHAR</td>
<td>VARCHAR (max 64KB) or STRING (can be read as either)</td>
</tr>
<tr>
<td>BINARY, VARBINARY, LONG VARBINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE if supported by your version of Hive, otherwise INT96 (can be read as TIMESTAMP)</td>
</tr>
<tr>
<td>TIMESTAMP, TIMESTAMPTZ</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>TIME, TIMEZ, INTERVAL</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Using the HCatalog Connector

The Vertica HCatalog Connector lets you access data stored in Apache's Hive data warehouse software the same way you access it within a native Vertica table.

If your files are in the Optimized Columnar Row (ORC) or Parquet format and do not use complex types, the HCatalog Connector creates an external table and uses the ORC or Parquet reader instead of using the Java SerDe. See Reading Hadoop Columnar File Formats for more information about these readers.

The HCatalog Connector performs predicate pushdown to improve query performance. Instead of reading all data across the network to evaluate a query, the HCatalog Connector moves the evaluation of predicates closer to the data. Predicate pushdown applies to Hive partition pruning, ORC stripe pruning, and Parquet row-group pruning. The HCatalog Connector supports predicate pushdown for the following predicates: >, >=, =, <=, <=, <.

Hive, HCatalog, and HiveServer2 Overview

There are several Hadoop components that you need to understand to use the HCatalog connector:

- Apache's Hive lets you query data stored in a Hadoop Distributed File System (HDFS) the same way you query data stored in a relational database. Behind the scenes, Hive uses a set of serializer and deserializer (SerDe) classes to extract data from files stored in HDFS and break it into columns and rows. Each SerDe handles data files in a specific format. For example, one SerDe extracts data from comma-separated data files while another interprets data stored in JSON format.

- Apache HCatalog is a component of the Hadoop ecosystem that makes Hive's metadata available to other Hadoop components (such as Pig).

- HiveServer2 makes HCatalog and Hive data available via JDBC. Through it, a client can make requests to retrieve data stored in Hive, as well as information about the Hive schema. HiveServer2 can use authorization services (Sentry or Ranger).

The Vertica HCatalog Connector lets you transparently access data that is available through HiveServer2. You use the connector to define a schema in Vertica that corresponds to a Hive database or schema. When you query data within this schema, the HCatalog Connector transparently extracts and formats the data from Hadoop into tabular data.
**Note:** You can use the WebHCat service instead of HiveServer2, but performance is usually better with HiveServer2. To use WebHCat, set the HCatalogConnectorUseHiveServer2 configuration parameter to 0. See Apache Hadoop Parameters. WebHCat does not support authorization services.

**HCatalog Connection Features**

The HCatalog Connector lets you query data stored in Hive using the Vertica native SQL syntax. Some of its main features are:

- The HCatalog Connector always reflects the current state of data stored in Hive.
- The HCatalog Connector uses the parallel nature of both Vertica and Hadoop to process Hive data. The result is that querying data through the HCatalog Connector is often faster than querying the data directly through Hive.
- Because Vertica performs the extraction and parsing of data, the HCatalog Connector does not significantly increase the load on your Hadoop cluster.
- The data you query through the HCatalog Connector can be used as if it were native Vertica data. For example, you can execute a query that joins data from a table in an HCatalog schema with a native table.

**HCatalog Connector Considerations**

There are a few things to keep in mind when using the HCatalog Connector:

- Hive's data is stored in flat files in a distributed filesystem, requiring it to be read and deserialized each time it is queried. This deserialization causes Hive performance to be much slower than that of Vertica. The HCatalog Connector has to perform the same process as Hive to read the data. Therefore, querying data stored in Hive using the HCatalog Connector is much slower than querying a native Vertica table. If you need to perform extensive analysis on data stored in Hive, you should consider loading it into Vertica. Vertica optimization often makes querying data through the HCatalog Connector faster than directly querying it through Hive.
- If Hive uses Kerberos security, the HCatalog Connector uses the querying user's credentials in queries by default. If Hive uses Sentry or Ranger to enforce security, then you must either
disable this behavior in Vertica by setting EnableHCatImpersonation to 0 or grant users access to the underlying data in HDFS. (Sentry supports ACL synchronization to automatically grant access.) See Configuring Kerberos.

- Hive supports complex data types such as lists, maps, and structs that Vertica does not support. Columns containing these data types are converted to a JSON representation of the data type and stored as a VARCHAR. See Data Type Conversions from Hive to Vertica.

**Note:** The HCatalog Connector is read-only. It cannot insert data into Hive.

## How the HCatalog Connector Works

When planning a query that accesses data from a Hive table, the Vertica HCatalog Connector on the initiator node contacts HiveServer2 (or WebHCat) in your Hadoop cluster to determine if the table exists. If it does, the connector retrieves the table’s metadata from the metastore database so the query planning can continue. When the query executes, all nodes in the Vertica cluster directly retrieve the data necessary for completing the query from HDFS. They then use the Hive SerDe classes to extract the data so the query can execute. When accessing data in ORC or Parquet format, the HCatalog Connector uses the readers for these formats instead of the Hive SerDe classes.
This approach takes advantage of the parallel nature of both Vertica and Hadoop. In addition, by performing the retrieval and extraction of data directly, the HCatalog Connector reduces the impact of the query on the Hadoop cluster.

**HCatalog Connector Requirements**

Before you can use the HCatalog Connector, both your Vertica and Hadoop installations must meet the following requirements.

**Vertica Requirements**

All of the nodes in your cluster must have a Java Virtual Machine (JVM) installed. You must use the same Java version that the Hadoop cluster uses. See Installing the Java Runtime on Your Vertica Cluster.

You must also add certain libraries distributed with Hadoop and Hive to your Vertica installation directory. See Configuring Vertica for HCatalog.

**Hadoop Requirements**

Your Hadoop cluster must meet several requirements to operate correctly with the Vertica Connector for HCatalog:

- It must have Hive, HiveServer2, and HCatalog installed and running. See Apache's HCatalog page for more information.

- The HiveServer2 server and all of the HDFS nodes that store HCatalog data must be directly accessible from all of the hosts in your Vertica database. Verify that any firewall separating the Hadoop cluster and the Vertica cluster will pass HiveServer2, metastore database, and HDFS traffic.

- The data that you want to query must be in an internal or external Hive table.

- When using the ORC and Parquet readers through the HCatalog Connector, partitioned data must be located in the default file location. If you have moved partitions, set the Vertica configuration parameters HCatalogConnectorUseORCReader and HCatalogConnectorUseParquetReader to 0 to use the Java UDx for these formats. Querying ORC and Parquet data through the Java UDx is slower than using the built-in readers.
If a table you want to query uses a non-standard SerDe, you must install the SerDe's classes on your Vertica cluster before you can query the data. See Using Nonstandard SerDes.

Installing the Java Runtime on Your Vertica Cluster

The HCatalog Connector requires a 64-bit Java Virtual Machine (JVM). The JVM must support Java 6 or later, and must be the same version as the one installed on your Hadoop nodes.

Note: If your Vertica cluster is configured to execute User Defined Extensions (UDxs) written in Java, it already has a correctly-configured JVM installed. See Developing User-Defined Extensions (UDxs) in Extending Vertica for more information.

Installing Java on your Vertica cluster is a two-step process:

1. Install a Java runtime on all of the hosts in your cluster.
2. Set the JavaBinaryForUDx configuration parameter to tell Vertica the location of the Java executable.

Installing a Java Runtime

For Java-based features, Vertica requires a 64-bit Java 6 (Java version 1.6) or later Java runtime. Vertica supports runtimes from either Oracle or OpenJDK. You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE.

Many Linux distributions include a package for the OpenJDK runtime. See your Linux distribution's documentation for information about installing and configuring OpenJDK.

To install the Oracle Java runtime, see the Java Standard Edition (SE) Download Page. You usually run the installation package as root in order to install it. See the download page for instructions.

Once you have installed a JVM on each host, ensure that the java command is in the search path and calls the correct JVM by running the command:

```
$ java -version
```

This command should print something similar to:
Note: Any previously installed Java VM on your hosts may interfere with a newly installed Java runtime. See your Linux distribution's documentation for instructions on configuring which JVM is the default. Unless absolutely required, you should uninstall any incompatible version of Java before installing the Java 6 or Java 7 runtime.

Setting the JavaBinaryForUDx Configuration Parameter

The JavaBinaryForUDx configuration parameter tells Vertica where to look for the JRE to execute Java UDxs. After you have installed the JRE on all of the nodes in your cluster, set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example /usr/bin/java). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

```
$ which java
/usr/bin/java
```

If the java command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. Suppose you installed the JRE in /usr/java/default (which is where the installation package supplied by Oracle installs the Java 1.6 JRE). In this case the Java executable is /usr/java/default/bin/java.

You set the configuration parameter by executing the following statement as a database superuser:

```
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
```

See ALTER DATABASE for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the CONFIGURATION_PARAMETERS system table:

```
=> \x
Expanded display is on.
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'JavaBinaryForUDx';
-[ RECORD 1 ]-----------------------------------------------------------------------------------------------------------
  node_name | ALL
  parameter_name | JavaBinaryForUDx
  current_value | /usr/bin/java
  default_value |
  change_under_support_guidance | f
```
Once you have set the configuration parameter, Vertica can find the Java executable on each node in your cluster.

**Note:** Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the Java executable is installed in the same path on all of the hosts in the cluster.

## Configuring Vertica for HCatalog

Before you can use the HCatalog Connector, you must add certain Hadoop and Hive libraries to your Vertica installation. You must also copy the Hadoop configuration files that specify various connection properties. Vertica uses the values in those configuration files to make its own connections to Hadoop.

You need only make these changes on one node in your cluster. After you do this you can install the HCatalog connector.

## Copy Hadoop Libraries and Configuration Files

Vertica provides a tool, hcatUtil, to collect the required files from Hadoop. This tool copies selected libraries and XML configuration files from your Hadoop cluster to your Vertica cluster. This tool might also need access to additional libraries:

- If you plan to use Hive to query files that use Snappy compression, you need access to the Snappy native libraries, libhadoop*.so and libsnappy*.so.

- If you plan to use Hive to query files that use LZO compression, you need access to the hadoop-lzo-*.jar and libgplcompression.so* libraries. In core-site.xml you must also edit the io.compression.codecs property to include com.hadoop.compression.lzo.LzopCodec.

- If you plan to use a JSON SerDe with a Hive table, you need access to its library. This is the same library that you used to configure Hive; for example:

  ```bash
  hive> add jar /home/release/json-serde-1.3-jar-with-dependencies.jar;
  ```
hive> create external table nationjson (id int, name string, rank int, text string) ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' LOCATION '/user/release/vt/nationjson';

- If you are using any other libraries that are not standard across all supported Hadoop versions, you need access to those libraries.

If any of these cases applies to you, do one of the following:

- Include the path(s) in the path you specify as the value of --hcatLibPath, or
- Copy the file(s) to a directory already on that path.

If Vertica is not co-located on a Hadoop node, you should do the following:

1. Copy /opt/vertica/packages/hcat/tools/hcatUtil to a Hadoop node and run it there, specifying a temporary output directory. Your Hadoop, HIVE, and HCatalog lib paths might be different. In newer versions of Hadoop the HCatalog directory is usually a subdirectory under the HIVE directory, and Cloudera creates a new directory for each revision of the configuration files. Use the values from your environment in the following command:

   ```bash
   hcatUtil --copyJars
   --hadoopHiveHome="$HADOOP_HOME/lib;$HIVE_HOME/lib;/hcatalog/dist/share"
   --hadoopHiveConfPath="$HADOOP_CONF_DIR;$HIVE_CONF_DIR;$WEBHCAT_CONF_DIR"
   --hcatLibPath="/tmp/hadoop-files"
   ```

2. Verify that all necessary files were copied:

   ```bash
   hcatUtil --verifyJars --hcatLibPath=/tmp/hadoop-files
   ```

3. Copy that output directory (/tmp/hadoop-files, in this example) to /opt/vertica/packages/hcat/lib on the Vertica node you will connect to when installing the HCatalog connector. If you are updating a Vertica cluster to use a new Hadoop cluster (or a new version of Hadoop), first remove all JAR files in /opt/vertica/packages/hcat/lib except vertica-hcatalogudl.jar.

4. Verify that all necessary files were copied:

   ```bash
   hcatUtil --verifyJars --hcatLibPath=/opt/vertica/packages/hcat
   ```

If Vertica is co-located on some or all Hadoop nodes, you can do this in one step on a shared node. Your Hadoop, HIVE, and HCatalog lib paths might be different; use the values from your environment in the following command:
hcatUtil --copyJars
  --hadoopHiveHome="$HADOOP_HOME/lib;$HIVE_HOME/lib;/hcatalog/dist/share"
  --hadoopHiveConfPath="$HADOOP_CONF_DIR;$HIVE_CONF_DIR;$WEBHCAT_CONF_DIR"
  --hcatLibPath="/opt/vertica/packages/hcat/lib"

The hcatUtil script has the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c, --copyJars</td>
<td>Copy the required JAR files from hadoopHiveHome and configuration files from hadoopHiveConfPath.</td>
</tr>
<tr>
<td>-v, --verifyJars</td>
<td>Verify that the required files are present in hcatLibPath. Check the output of hcatUtil for error and warning messages.</td>
</tr>
</tbody>
</table>
| --hadoopHiveHome="value1:value2;..." | Paths to the Hadoop, Hive, and HCatalog home directories. Separate directories by semicolons (;). Enclose paths in double quotes.  
  **Note:** Always place $HADOOP_HOME on the path before $HIVE_HOME. In some Hadoop distributions, these two directories contain different versions of the same library. |
| --hadoopHiveConfPath="value1:value2;..." | Paths of the following configuration files:  
  - hive-site.xml  
  - core-site.xml  
  - yarn-site.xml  
  - webhcat-site.xml (optional with the default configuration; required if you use WebHCat instead of HiveServer2)  
  - hdfs-site.xml  
  Separate directories by semicolons (;). Enclose paths in double quotes.  
  In previous releases of Vertica this parameter was optional under some conditions. It is now required. |
| --hcatLibPath="value" | Output path for the libraries and configuration files. On a Vertica node, use /opt/vertica/packages/hcat/lib. If you have previously run hcatUtil with a different version of Hadoop, first remove the old JAR files from the output directory (all except vertica-hcatalogudl.jar). |
After you have copied the files and verified them, install the HCatalog connector.

Install the HCatalog Connector

On the same node where you copied the files from hcatUtil, install the HCatalog connector by running the install.sql script. This script resides in the ddl/ folder under your HCatalog connector installation path. This script creates the library and VHCatSource and VHCatParser.

Note: The data that was copied using hcatUtil is now stored in the database. If you change any of those values in Hadoop, you need to rerun hcatUtil and install.sql. The following statement returns the names of the libraries and configuration files currently being used:

```sql
=> SELECT dependencies FROM user_libraries WHERE lib_name='VHCatalogLib';
```

Now you can create HCatalog schema parameters, which point to your existing Hadoop services, as described in Defining a Schema Using the HCatalog Connector.

Upgrading to a New Version of Vertica

After upgrading to a new version of Vertica, perform the following steps:

1. Uninstall the HCatalog Connector using the uninstall.sql script. This script resides in the ddl/ folder under your HCatalog connector installation path.
2. Delete the contents of the hcatLibPath directory except for vertica-hcatalogudl.jar.
3. Rerun hcatUtil.
4. Reinstall the HCatalog Connector using the install.sql script.

For more information about upgrading Vertica, see Upgrade Vertica.

Additional Options for Hadoop Columnar File Formats

When reading Hadoop columnar file formats (ORC or Parquet), the HCatalog Connector attempts to use the built-in readers. When doing so, it uses the hdfs scheme by default. In order to use the hdfs scheme, you must perform the configuration described in Configuring the hdfs Scheme.
To have the HCatalog Connector use the webhdfs scheme instead, use `ALTER DATABASE` to set HCatalogConnectorUseLibHDFSPP to 0.

### Defining a Schema Using the HCatalog Connector

After you set up the HCatalog Connector, you can use it to define a schema in your Vertica database to access the tables in a Hive database. You define the schema using the `CREATE HCATALOG SCHEMA` statement.

When creating the schema, you must supply the name of the schema to define in Vertica. Other parameters are optional. If you do not supply a value, Vertica uses default values. Vertica reads some default values from the HDFS configuration files; see [Configuration Parameters](#). To create the schema, you must have read access to all Hive data. Verify that the user creating the schema has been granted access, either directly or through an authorization service such as Sentry or Ranger. The dbadmin user has no automatic special privileges.

After you create the schema, you can change many parameters using the `ALTER HCATALOG SCHEMA` statement.

After you define the schema, you can query the data in the Hive data warehouse in the same way you query a native Vertica table. The following example demonstrates creating an HCatalog schema and then querying several system tables to examine the contents of the new schema. See [Viewing Hive Schema and Table Metadata](#) for more information about these tables.

```sql
=> CREATE HCATALOG SCHEMA hcat WITH HOSTNAME='hcathost' PORT=9083
     HCATALOG_SCHEMA='default' HIVESERVER2_HOSTNAME='hs.example.com'
     SSL_CONFIG='/etc/hadoop/conf/ssl-client.xml' HCATALOG_USER='admin';
CREATE SCHEMA
=> \x
Expanded display is on.

=> SELECT * FROM v_catalog.hcatalog_schemata;
- [ RECORD 1 ]----------------------------------------
schema_id | 45035996273748224
schema_name | hcat
schema_owner_id | 45035996273704962
schema_owner | admin
create_time | 2017-12-05 14:43:03.353404-05
hostname | hcathost
port | -1
hiveserver2_hostname | hs.example.com
webservice_hostname | 
webservice_port | 
webhdfs_address | hs.example.com:50070
```
Configuration Parameters

The HCatalog Connector uses the following values from the Hadoop configuration files if you do not override them when creating the schema.

<table>
<thead>
<tr>
<th>File</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive-site.xml</td>
<td>hive.server2.thrift.bind.host (used for HIVESERVER2_HOSTNAME)</td>
</tr>
<tr>
<td></td>
<td>hive.server2.thrift.port</td>
</tr>
<tr>
<td></td>
<td>hive.server2.transport.mode</td>
</tr>
<tr>
<td></td>
<td>hive.server2.authentication</td>
</tr>
<tr>
<td></td>
<td>hive.server2.authentication.kerberos.principal</td>
</tr>
</tbody>
</table>
Using the HCatalog Connector with WebHCat

By default the HCatalog Connector uses HiveServer2 to access Hive data. If you are instead using WebHCat, set the HCatalogConnectorUseHiveServer2 configuration parameter to 0 before creating the schema as in the following example.

```sql
=> ALTER DATABASE mydb SET HCatalogConnectorUseHiveServer2 = 0;
=> CREATE HCATALOG SCHEMA hcat WITH WEBSERVICE_HOSTNAME='webhcat.example.com';
```

If you have previously used WebHCat, you can switch to using HiveServer2 by setting the configuration parameter to 1 and using `ALTER HCATALOG SCHEMA` to set `HIVESERVER2_HOSTNAME`. You do not need to remove the WebHCat values; the HCatalog Connector uses the value of HCatalogConnectorUseHiveServer2 to determine which parameters to use.

Querying Hive Tables Using HCatalog Connector

Once you have defined the HCatalog schema, you can query data from the Hive database by using the schema name in your query.
Since the tables you access through the HCatalog Connector act like Vertica tables, you can perform operations that use both Hive data and native Vertica data, such as a join:

```
=> SELECT * from hcat.messages limit 10;
messageid | userid | time | message
----------|--------|------|---------------------------
1 | npFQ1ayhi | 2013-10-29 00:10:43 | hymenaeos cursus lorem Suspendis
2 | N7svORIoZ | 2013-10-29 00:21:27 | Fusce ad sem vehicula morbi
3 | 4VvzN3d | 2013-10-29 00:32:11 | porta Vivamus condimentum
4 | he0jkmTmc | 2013-10-29 00:42:55 | lectus quis imperdiet
5 | coROws3OF | 2013-10-29 00:53:39 | sit eleifend tempus a aliquam mauri
6 | oDRPi1 | 2013-10-29 01:04:23 | risus facilisis sollicitudin sceler
7 | AU7a9Kp | 2013-10-29 01:15:07 | turpis vehicula tortor
8 | ZJWg185DkZ | 2013-10-29 01:25:51 | sapien adipiscing eget Aliquam tor
9 | E7ipAsYC3 | 2013-10-29 01:36:35 | varius Cum iaculis metus
10 | kStCv | 2013-10-29 01:47:19 | aliquam libero nascetur Cum mal
(10 rows)
```

Viewing Hive Schema and Table Metadata

When using Hive, you access metadata about schemas and tables by executing statements written in HiveQL (Hive’s version of SQL) such as `SHOW TABLES`. When using the HCatalog Connector, you can get metadata about the tables in the Hive database through several Vertica system tables.

There are four system tables that contain metadata about the tables accessible through the HCatalog Connector:

- `HCATALOG_SCHEMATA` lists all of the schemas that have been defined using the HCatalog Connector.

```
- **HCATALOG_TABLE_LIST** contains an overview of all of the tables available from all schemas defined using the HCatalog Connector. This table only shows the tables that the user querying the table can access. The information in this table is retrieved using a single call to HiveServer2 for each schema defined using the HCatalog Connector, which means there is a little overhead when querying this table.

- **HCATALOG_TABLES** contains more in-depth information than HCATALOG_TABLE_LIST.

- **HCATALOG_COLUMNS** lists metadata about all of the columns in all of the tables available through the HCatalog Connector. As for HCATALOG_TABLES, querying this table results in one call to HiveServer2 per table, and therefore can take a while to complete.

The following example demonstrates querying the system tables containing metadata for the tables available through the HCatalog Connector.

```sql
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost'
-> HCATALOG_SCHEMA='default' HCATALOG_DB='default' HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT * FROM HCATALOG_SCHEMA;
- [ RECORD 1 ]-----------------------------------------------
schema_id | 45035996273864536
schema_name | hcat
schema_owner_id | 45035996273704962
schema_owner | dbadmin
create_time | 2013-11-05 10:19:54.70965-05
hostname | hcathost
port | 9083
webservice_hostname | hcathost
webservice_port | 50111
hcatalog_schema_name | default
hcatalog_user_name | hcatuser
metastore_db_name | hivemetastoredb

=> SELECT * FROM HCATALOG_TABLE_LIST;
- [ RECORD 1 ]-----------------
table_schema_id | 45035996273864536
table_schema | hcat
hcatalog_schema | default
table_name | hcatalogtypes
hcatalog_user_name | hcatuser
- [ RECORD 2 ]-----------------
table_schema_id | 45035996273864536
table_schema | hcat
hcatalog_schema | default
table_name | tweets
hcatalog_user_name | hcatuser
- [ RECORD 3 ]-----------------
table_schema_id | 45035996273864536
table_schema | hcat
hcatalog_schema | default
table_name | messages
hcatalog_user_name | hcatuser
- [ RECORD 4 ]-----------------
table_schema_id | 45035996273864536
```
<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>msgjson</td>
</tr>
<tr>
<td>hcatalog_user_name</td>
<td>hcatuser</td>
</tr>
</tbody>
</table>

Get detailed description of a specific table

```sql
SELECT * FROM HCATALOG_TABLES WHERE table_name = 'msgjson';
```

<table>
<thead>
<tr>
<th>table_schema_id</th>
<th>45035996273864536</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_schema</td>
<td>hcat</td>
</tr>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>msgjson</td>
</tr>
<tr>
<td>hcatalog_user_name</td>
<td>hcatuser</td>
</tr>
<tr>
<td>min_file_size_bytes</td>
<td></td>
</tr>
<tr>
<td>total_number_files</td>
<td>10</td>
</tr>
<tr>
<td>location</td>
<td>hdfs://hive.example.com:8020/user/exampleuser/msgjson</td>
</tr>
<tr>
<td>last_update_time</td>
<td></td>
</tr>
<tr>
<td>output_format</td>
<td>org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat</td>
</tr>
<tr>
<td>last_access_time</td>
<td></td>
</tr>
<tr>
<td>max_file_size_bytes</td>
<td>f</td>
</tr>
<tr>
<td>is_partitioned</td>
<td></td>
</tr>
<tr>
<td>partition_expression</td>
<td></td>
</tr>
<tr>
<td>table_owner</td>
<td></td>
</tr>
<tr>
<td>input_format</td>
<td>org.apache.hadoop.mapred.TextInputFormat</td>
</tr>
<tr>
<td>total_file_size_bytes</td>
<td>453534</td>
</tr>
<tr>
<td>hcatalog_group</td>
<td></td>
</tr>
<tr>
<td>permission</td>
<td></td>
</tr>
</tbody>
</table>

Get list of columns in a specific table

```sql
SELECT * FROM HCATALOG_COLUMNS WHERE table_name = 'hcatalogtypes'
ORDER BY ordinal_position;
```

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>intcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>int</td>
</tr>
<tr>
<td>data_type</td>
<td>int</td>
</tr>
<tr>
<td>data_type_id</td>
<td>6</td>
</tr>
<tr>
<td>data_type_length</td>
<td>8</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>floatcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>float</td>
</tr>
<tr>
<td>data_type</td>
<td>float</td>
</tr>
<tr>
<td>data_type_id</td>
<td>7</td>
</tr>
<tr>
<td>data_type_length</td>
<td>8</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>---</td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>2</td>
</tr>
</tbody>
</table>

- [ RECORD 3 ]----------------------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>doublecol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>double</td>
</tr>
<tr>
<td>data_type</td>
<td>float</td>
</tr>
<tr>
<td>data_type_id</td>
<td>7</td>
</tr>
<tr>
<td>data_type_length</td>
<td>8</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>3</td>
</tr>
</tbody>
</table>

- [ RECORD 4 ]----------------------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>charcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>string</td>
</tr>
<tr>
<td>data_type</td>
<td>varchar(65000)</td>
</tr>
<tr>
<td>data_type_id</td>
<td>9</td>
</tr>
<tr>
<td>data_type_length</td>
<td>65000</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>4</td>
</tr>
</tbody>
</table>

- [ RECORD 5 ]----------------------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>varcharcharcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>string</td>
</tr>
<tr>
<td>data_type</td>
<td>varchar(65000)</td>
</tr>
<tr>
<td>data_type_id</td>
<td>9</td>
</tr>
<tr>
<td>data_type_length</td>
<td>65000</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>5</td>
</tr>
</tbody>
</table>

- [ RECORD 6 ]----------------------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>boolcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>boolean</td>
</tr>
<tr>
<td>data_type</td>
<td>boolean</td>
</tr>
<tr>
<td>data_type_id</td>
<td>5</td>
</tr>
<tr>
<td>data_type_length</td>
<td>1</td>
</tr>
<tr>
<td>------------------</td>
<td>---</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td></td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>6</td>
</tr>
</tbody>
</table>

- [RECORD 7]-----------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>timestampcol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>string</td>
</tr>
<tr>
<td>data_type</td>
<td>varchar(65000)</td>
</tr>
<tr>
<td>data_type_id</td>
<td>9</td>
</tr>
<tr>
<td>data_type_length</td>
<td>65000</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td>65000</td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>7</td>
</tr>
</tbody>
</table>

- [RECORD 8]-----------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>varbincol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>binary</td>
</tr>
<tr>
<td>data_type</td>
<td>varbinary(65000)</td>
</tr>
<tr>
<td>data_type_id</td>
<td>17</td>
</tr>
<tr>
<td>data_type_length</td>
<td>65000</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td>65000</td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>8</td>
</tr>
</tbody>
</table>

- [RECORD 9]-----------------------------

<table>
<thead>
<tr>
<th>table_schema</th>
<th>hcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcatalog_schema</td>
<td>default</td>
</tr>
<tr>
<td>table_name</td>
<td>hcatalogtypes</td>
</tr>
<tr>
<td>is_partition_column</td>
<td>f</td>
</tr>
<tr>
<td>column_name</td>
<td>bincol</td>
</tr>
<tr>
<td>hcatalog_data_type</td>
<td>binary</td>
</tr>
<tr>
<td>data_type</td>
<td>varbinary(65000)</td>
</tr>
<tr>
<td>data_type_id</td>
<td>17</td>
</tr>
<tr>
<td>data_type_length</td>
<td>65000</td>
</tr>
<tr>
<td>character_maximum_length</td>
<td>65000</td>
</tr>
<tr>
<td>numeric_precision</td>
<td></td>
</tr>
<tr>
<td>numeric_scale</td>
<td></td>
</tr>
<tr>
<td>datetime_precision</td>
<td></td>
</tr>
<tr>
<td>interval_precision</td>
<td></td>
</tr>
<tr>
<td>ordinal_position</td>
<td>9</td>
</tr>
</tbody>
</table>
Synchronizing an HCatalog Schema or Table With a Local Schema or Table

Querying data from an HCatalog schema can be slow due to Hive performance issues. This slow performance can be especially annoying when you want to examine the structure of the tables in the Hive database. Getting this information from Hive requires you to query the HCatalog schema's metadata using the HCatalog Connector.

To avoid this performance problem you can use the SYNC_WITH_HCATALOG_SCHEMA function to create a snapshot of the HCatalog schema's metadata within a Vertica schema. You supply this function with the name of a pre-existing Vertica schema, typically the one created through CREATE HCATALOG SCHEMA, and a Hive schema available through the HCatalog Connector. You must have permission both in Vertica to write the data and in Hive and HDFS to read it.

**Note:** To synchronize a schema, you must have read permission for the underlying files in HDFS. If Hive uses Sentry to manage authorization, then you can use ACL synchronization to manage HDFS access. Otherwise, the user of this function must have read access in HDFS.

The function creates a set of external tables within the Vertica schema that you can then use to examine the structure of the tables in the Hive database. Because the metadata in the Vertica schema is local, query planning is much faster. You can also use standard Vertica statements and system-table queries to examine the structure of Hive tables in the HCatalog schema.

**Caution:** The SYNC_WITH_HCATALOG_SCHEMA function overwrites tables in the Vertica schema whose names match a table in the HCatalog schema. Do not use the Vertica schema to store other data.

When SYNC_WITH_HCATALOG_SCHEMA creates tables in Vertica, it matches Hive's STRING and BINARY types to Vertica's VARCHAR(65000) and VARBINARY(65000) types. You might want to change these lengths, using ALTER TABLE SET DATA TYPE, in two cases:

- If the value in Hive is larger than 65000 bytes, increase the size and use LONG VARCHAR or LONG VARBINARY to avoid data truncation. If a Hive string uses multi-byte encodings, you must increase the size in Vertica to avoid data truncation. This step is needed because Hive counts string length in characters while Vertica counts it in bytes.

- If the value in Hive is much smaller than 65000 bytes, reduce the size to conserve memory in Vertica.
The Vertica schema is just a snapshot of the HCatalog schema's metadata. Vertica does not synchronize later changes to the HCatalog schema with the local schema after you call SYNC_WITH_HCATALOG_SCHEMA. You can call the function again to re-synchronize the local schema to the HCatalog schema. If you altered column data types, you will need to repeat those changes because the function creates new external tables.

By default, SYNC_WITH_HCATALOG_SCHEMA does not drop tables that appear in the local schema that do not appear in the HCatalog schema. Thus, after the function call the local schema does not reflect tables that have been dropped in the Hive database since the previous call. You can change this behavior by supplying the optional third Boolean argument that tells the function to drop any table in the local schema that does not correspond to a table in the HCatalog schema.

Instead of synchronizing the entire schema, you can synchronize individual tables by using SYNC_WITH_HCATALOG_SCHEMA_TABLE. If the table already exists in Vertica the function overwrites it. If the table is not found in the HCatalog schema, this function returns an error. In all other respects this function behaves in the same way as SYNC_WITH_HCATALOG_SCHEMA.

If you change the settings of any HCatalog Connector configuration parameters (Apache Hadoop Parameters), you must call this function again.

### Examples

The following example demonstrates calling SYNC_WITH_HCATALOG_SCHEMA to synchronize the HCatalog schema in Vertica with the metadata in Hive. Because it synchronizes the HCatalog schema directly, instead of synchronizing another schema with the HCatalog schema, both arguments are the same.

```sql
=> CREATE HCATALOG SCHEMA hcat WITH hostname='hcathost' HCATALOG_SCHEMA='default'
       HCATALOG_USER='hcatuser';
CREATE SCHEMA
=> SELECT sync_with_hcatalog_schema('hcat', 'hcat');
sync_with_hcatalog_schema
-----------------------------------------------
Schema hcat synchronized with hcat
  tables in hcat = 56
  tables altered in hcat = 0
  tables created in hcat = 56
  stale tables in hcat = 0
  table changes errored in hcat = 0
(1 row)
=> -- Use vsq1's \d command to describe a table in the synced schema
=> \d hcat.messages
List of Fields by Tables
  Schema | Table | Column | Type | Size | Default | Not Null | Primary Key | Foreign Key
```
This example shows synchronizing with a schema created using CREATE HCATALOG SCHEMA. Synchronizing with a schema created using CREATE SCHEMA is also supported.

You can query tables in the local schema that you synchronized with an HCatalog schema. However, querying tables in a synchronized schema isn't much faster than directly querying the HCatalog schema, because SYNC_WITH_HCATALOG_SCHEMA only duplicates the HCatalog schema's metadata. The data in the table is still retrieved using the HCatalog Connector.

### Data Type Conversions from Hive to Vertica

The data types recognized by Hive differ from the data types recognized by Vertica. The following table lists how the HCatalog Connector converts Hive data types into data types compatible with Vertica.

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>Vertica Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT (1-byte)</td>
<td>TINYINT (8-bytes)</td>
</tr>
<tr>
<td>SMALLINT (2-bytes)</td>
<td>SMALLINT (8-bytes)</td>
</tr>
<tr>
<td>INT (4-bytes)</td>
<td>INT (8-bytes)</td>
</tr>
<tr>
<td>BIGINT (8-bytes)</td>
<td>BIGINT (8-bytes)</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>FLOAT (4-bytes)</td>
<td>FLOAT (8-bytes)</td>
</tr>
<tr>
<td>DECIMAL (precision, scale)</td>
<td>DECIMAL (precision, scale)</td>
</tr>
<tr>
<td>DOUBLE (8-bytes)</td>
<td>DOUBLE PRECISION (8-bytes)</td>
</tr>
<tr>
<td>CHAR (length in characters)</td>
<td>CHAR (length in bytes)</td>
</tr>
<tr>
<td>VARCHAR (length in characters)</td>
<td>VARCHAR (length in bytes), if length &lt;= 65000 LONG VARCHAR (length in bytes), if length &gt; 65000</td>
</tr>
<tr>
<td>Hive Data Type</td>
<td>Vertica Data Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STRING (2 GB max)</td>
<td>VARCHAR (65000)</td>
</tr>
<tr>
<td>BINARY (2 GB max)</td>
<td>VARBINARY (65000)</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>LIST/ARRAY</td>
<td>VARCHAR (65000) containing a JSON-format representation of the list.</td>
</tr>
<tr>
<td>MAP</td>
<td>VARCHAR (65000) containing a JSON-format representation of the map.</td>
</tr>
<tr>
<td>STRUCT</td>
<td>VARCHAR (65000) containing a JSON-format representation of the struct.</td>
</tr>
</tbody>
</table>

### Data-Width Handling Differences Between Hive and Vertica

The HCatalog Connector relies on Hive SerDe classes to extract data from files on HDFS. Therefore, the data read from these files are subject to Hive's data width restrictions. For example, suppose the SerDe parses a value for an INT column into a value that is greater than $2^{32}-1$ (the maximum value for a 32-bit integer). In this case, the value is rejected even if it would fit into a Vertica's 64-bit INTEGER column because it cannot fit into Hive's 32-bit INT.

Hive measures CHAR and VARCHAR length in characters and Vertica measures them in bytes. Therefore, if multi-byte encodings are being used (like Unicode), text might be truncated in Vertica.

Once the value has been parsed and converted to a Vertica data type, it is treated as native data. This treatment can result in some confusion when comparing the results of an identical query run in Hive and in Vertica. For example, if your query adds two INT values that result in a value that is larger than $2^{32}-1$, the value overflows its 32-bit INT data type, causing Hive to return an error. When running the same query with the same data in Vertica using the HCatalog Connector, the value will probably still fit within Vertica's 64-int value. Thus the addition is successful and returns a value.
Using Nonstandard SerDes

Hive stores its data in unstructured flat files located in the Hadoop Distributed File System (HDFS). When you execute a Hive query, it uses a set of serializer and deserializer (SerDe) classes to extract data from these flat files and organize it into a relational database table. For Hive to be able to extract data from a file, it must have a SerDe that can parse the data the file contains. When you create a table in Hive, you can select the SerDe to be used for the table's data.

Hive has a set of standard SerDes that handle data in several formats such as delimited data and data extracted using regular expressions. You can also use third-party or custom-defined SerDes that allow Hive to process data stored in other file formats. For example, some commonly-used third-party SerDes handle data stored in JSON format.

The HCatalog Connector directly fetches file segments from HDFS and uses Hive's SerDes classes to extract data from them. The Connector includes all Hive's standard SerDes classes, so it can process data stored in any file that Hive natively supports. If you want to query data from a Hive table that uses a custom SerDe, you must first install the SerDe classes on the Vertica cluster.

Determining Which SerDe You Need

If you have access to the Hive command line, you can determine which SerDe a table uses by using Hive's SHOW CREATE TABLE statement. This statement shows the HiveQL statement needed to recreate the table. For example:

```
hive> SHOW CREATE TABLE msgjson;
OK
CREATE EXTERNAL TABLE msgjson(
  messageid int COMMENT 'from deserializer',
  userid string COMMENT 'from deserializer',
  time string COMMENT 'from deserializer',
  message string COMMENT 'from deserializer')
ROW FORMAT SERDE
  'org.apache.hadoop.hive.contrib.serde2.JsonSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://hivehost.example.com:8020/user/exampleuser/msgjson'
TBLPROPERTIES(
  'transient_lastDdlTime'='1384194521')
Time taken: 0.167 seconds
```
In the example, **ROW FORMAT SERDE** indicates that a special SerDe is used to parse the data files. The next row shows that the class for the SerDe is named `org.apache.hadoop.hive.contrib.serde2.JsonSerde`. You must provide the HCatalog Connector with a copy of this SerDe class so that it can read the data from this table.

You can also find out which SerDe class you need by querying the table that uses the custom SerDe. The query will fail with an error message that contains the class name of the SerDe needed to parse the data in the table. In the following example, the portion of the error message that names the missing SerDe class is in bold.

```sql
=> SELECT * FROM hcat.jsontable;
ERROR 3399: Failure in U Dx RPC call InvokePlanUDL(): Error in User Defined Object [VHCatSource], error code: 0
```

### Installing the SerDe on the Vertica Cluster

You usually have two options to getting the SerDe class file the HCatalog Connector needs:

- Find the installation files for the SerDe, then copy those over to your Vertica cluster. For example, there are several third-party JSON SerDes available from sites like Google Code and GitHub. You may find the one that matches the file installed on your Hive cluster. If so, then download the package and copy it to your Vertica cluster.

- Directly copy the JAR files from a Hive server onto your Vertica cluster. The location for the SerDe JAR files depends on your Hive installation. On some systems, they may be located in `/usr/lib/hive/lib`.

Wherever you get the files, copy them into the `/opt/vertica/packages/hcat/lib` directory on every node in your Vertica cluster.

**Important:** If you add a new host to your Vertica cluster, remember to copy every custom SerDer JAR file to it.
Troubleshooting HCatalog Connector Problems

You may encounter the following issues when using the HCatalog Connector.

Connection Errors

When you use CREATE HCATALOG SCHEMA to create a new schema, the HCatalog Connector does not immediately attempt to connect to the HiveServer2 or metastore servers. Instead, when you execute a query using the schema or HCatalog-related system tables, the connector attempts to connect to and retrieve data from your Hadoop cluster.

The types of errors you get depend on which parameters are incorrect. Suppose you have incorrect parameters for the metastore database, but correct parameters for HiveServer2. In this case, HCatalog-related system table queries succeed, while queries on the HCatalog schema fail. The following example demonstrates creating an HCatalog schema with the correct default HiveServer2 information. However, the port number for the metastore database is incorrect.

```sql
=> CREATE HCATALOG SCHEMA hcat2 WITH hostname='hcathost'
  HCATALOG_SCHEMA='default' HCATALOG_USER='hive' PORT=1234;
CREATE SCHEMA
=> SELECT * FROM HCATALOG_TABLE_LIST;
-[: RECORD 1 ]-----------------------------
table_schema_id | 45035996273864536
table_schema | hcat2
hcatalog_schema | default
table_name | test
hcatalog_user_name | hive

=> SELECT * FROM hcat2.test;
ERROR 3399: Failure in UDX RPC call InvokePlanUDL(): Error in User Defined Object [VHCatSource], error code: 0
com.vertica.sdk.UdfException: Error message is [
org.apache.hcatalog.common.HCatException : 2004 : HCatOutputFormat not initialized, setOutput has to be called. Cause : java.io.IOException:MetaException(message:Could not connect to meta store using any of the URIs provided. Most recent failure: org.apache.thrift.transport.TTransportException: java.net.ConnectException:Connection refused
at org.apache.thrift.transport.TSocket.open(TSocket.java:185)
 at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.open(HiveMetaStoreClient.java:277)
```
To resolve these issues, you must drop and recreate the schema or alter the schema to correct the parameters. If you still have issues, determine whether there are connectivity issues between your Vertica cluster and your Hadoop cluster. Such issues can include a firewall that prevents one or more Vertica hosts from contacting the HiveServer2, metastore, or HDFS hosts.

**UDx Failure When Querying Data: Error 3399**

You might see an error message when querying data (as opposed to metadata like schema information). This might be accompanied by a ClassNotFoundException in the log. This can happen for the following reasons:

- You are not using the same version of Java on your Hadoop and Vertica nodes. In this case you need to change one of them to match the other.
- You have not used hcatUtil to copy all Hadoop and Hive libraries and configuration files to Vertica, or you ran hcatutil and then changed your version of Hadoop or Hive.
- You upgraded Vertica to a new version and did not rerun hcatutil and reinstall the HCatalog Connector.
- The version of Hadoop you are using relies on a third-party library that you must copy manually.
- You are reading files with LZO compression and have not copied the libraries or set the io.compression.codecs property in core-site.xml.
- You are reading Parquet data from Hive, and columns were added to the table after some data was already present in the table. Adding columns does not update existing data, and the ParquetSerDe provided by Hive and used by the HCatalog Connector does not handle this case. This error is due to a limitation in Hive and there is no workaround.
- The query is taking too long and is timing out. If this is a frequent problem, you can increase the value of the UDxFencedBlockTimeout configuration parameter. See General Parameters.

If you did not copy the libraries or configure LZO compression, follow the instructions in Configuring Vertica for HCatalog.

If the Hive jars that you copied from Hadoop are out of date, you might see an error message like the following:
This error usually signals a problem with hive-hcatalog-core . Make sure you have an up-to-date copy of this file. Remember that if you rerun hcatUtil you also need to re-create the HCatalog schema.

You might also see a different form of this error:

This error can be reported even if hcatUtil reports that your libraries are up to date. The javax.servlet.Filter class is in a library that some versions of Hadoop use but that is not usually part of the Hadoop installation directly. If you see an error mentioning this class, locate servlet-api-*.jar on a Hadoop node and copy it to the hcat/lib directory on all database nodes. If you cannot locate it on a Hadoop node, locate and download it from the Internet. (This case is rare.) The library version must be 2.3 or higher.

After you have copied the jar to the hcat/lib directory, reinstall the HCatalog connector as explained in Configuring Vertica for HCatalog.

Authentication Error When Querying Data

You might have successfully created a schema using CREATE HCATALOG SCHEMA but get errors at query time such as the following:

You might see this error if Hive uses an authorization service. If the permissions on the underlying files in HDFS match those in the authorization service, then Vertica must use user impersonation when accessing that data. To enable user impersonation, set the EnableHCatImpersonation configuration parameter to 1.

Vertica uses the database principal to access HDFS. Therefore, if EnableHCatImpersonation is 0, the Vertica database principal must have access to the data inside the hive warehouse on HDFS. If it does not, you might see the following error:
The URL scheme in this error message has been changed from hdfs to vertica_hdfs. This is an internal scheme and is not valid in URLs outside of Vertica. You cannot use this scheme when specifying paths in HDFS.

SerDe Errors

Errors can occur if you attempt to query a Hive table that uses a nonstandard SerDe. If you have not installed the SerDe JAR files on your Vertica cluster, you receive an error similar to the one in the following example:

```
=> SELECT * FROM hcat.jsontable;
ERROR 3399: Failure in U Dx RPC call InvokePlanUDL(): Error in User Defined Object [VHCatSource], error code: 0
HINT If error message is not descriptive or local, may be we cannot read metadata from hive metastore service thrift://hcathost:9083 or HDFS namenode (check UDxLogs/UDxFencedProcessesJava.log in the catalog directory for more information)
at com.vertica.hcatalogudl.HCatalogSplitsNoOpSourceFactory.plan(HCatalogSplitsNoOpSourceFactory.java:98)
at com.vertica.udxfence.UDxExecContext.planUDSource(UDxExecContext.java:898)
...
```

In the error message, you can see that the root cause is a missing SerDe class (shown in bold). To resolve this issue, install the SerDe class on your Vertica cluster. See Using Nonstandard SerDes for more information.

This error may occur intermittently if just one or a few hosts in your cluster do not have the SerDe class.

Differing Results Between Hive and Vertica Queries

Sometimes, running the same query on Hive and on Vertica through the HCatal og Connector can return different results. This discrepancy is often caused by the differences between the data types supported by Hive and Vertica. See Data Type Conversions from Hive to Vertica for more information about supported data types.
If Hive string values are being truncated in Vertica, this might be caused by multi-byte character encodings in Hive. Hive reports string length in characters, while Vertica records it in bytes. For a two-byte encoding such as Unicode, you need to double the column size in Vertica to avoid truncation.

Discrepancies can also occur if the Hive table uses partition columns of types other than string.

**HCatalog Connector Installation Fails on MapR**

If you mount a MapR file system as an NFS mount point and then install the HCatalog Connector, it could fail with a message like the following:

ROLLBACK 2929: Couldn't create new UDx side process, failed to get UDx side process info from zygote: Broken pipe

This might be accompanied by an error like the following in dbLog:

```java
java.io.IOException: Couldn't get lock for /home/dbadmin/node02_catalog/UDxLogs/UDxFencedProcessesJava.log
    at java.util.logging.FileHandler.openFiles(FileHandler.java:389)
    at java.util.logging.FileHandler.<init>(FileHandler.java:287)
    at com.vertica.udxfence.UDxLogger.setup(UDxLogger.java:78)
    at com.vertica.udxfence.UDxSideProcess.go(UDxSideProcess.java:75)
    ...
```

This error occurs if you locked your NFS mount point when creating it. Locking is the default. If you use the HCatalog Connector with MapR mounted as an NFS mount point, you must create the mount point with the `-o nolock` option. For example:

```bash
sudo mount -o nolock -t nfs MaprCLDBserviceHostname:/mapr/ClusterName/vertica/$(hostname -f)/vertica
```

You can use the HCatalog Connector with MapR without mounting the MapR file system. If you mount the MapR file system, you must do so without a lock.

**Preventing Excessive Query Delays**

Network issues or high system loads on the HiveServer2 server can cause long delays while querying a Hive database using the HCatalog Connector. While Vertica cannot resolve these issues, you can set parameters that limit how long Vertica waits before canceling a query on an HCatalog schema. You can set these parameters globally using Vertica configuration parameters. You can also set them for specific HCatalog schemas in the `CREATE HCATALOG SCHEMA` statement. These specific settings override the settings in the configuration parameters.
The HCatConnectionTimeout configuration parameter and the CREATE HCATALOG SCHEMA statement's HCATALOG_CONNECTION_TIMEOUT parameter control how many seconds the HCatalog Connector waits for a connection to the HiveServer2 server. A value of 0 (the default setting for the configuration parameter) means to wait indefinitely. If the server does not respond by the time this timeout elapses, the HCatalog Connector breaks the connection and cancels the query. If you find that some queries on an HCatalog schema pause excessively, try setting this parameter to a timeout value, so the query does not hang indefinitely.

The HCatSlowTransferTime configuration parameter and the CREATE HCATALOG SCHEMA statement's HCATALOG_SLOW_TRANSFER_TIME parameter specify how long the HCatlog Connector waits for data after making a successful connection to the server. After the specified time has elapsed, the HCatalog Connector determines whether the data transfer rate from the server is at least the value set in the HCatSlowTransferLimit configuration parameter (or by the CREATE HCATALOG SCHEMA statement's HCATALOG_SLOW_TRANSFER_LIMIT parameter). If it is not, then the HCatalog Connector terminates the connection and cancels the query.

You can set these parameters to cancel queries that run very slowly but do eventually complete. However, query delays are usually caused by a slow connection rather than a problem establishing the connection. Therefore, try adjusting the slow transfer rate settings first. If you find the cause of the issue is connections that never complete, you can alternately adjust the Linux TCP socket timeouts to a suitable value instead of relying solely on the HCatConnectionTimeout parameter.
Using HDFS Storage Locations

Vertica stores data in its native format, ROS, in storage locations. You can place storage locations on the local Linux file system or in HDFS. If you are using Premium Edition, you typically use HDFS storage locations for lower-priority data. Doing so frees space on your Vertica cluster for higher-priority data. If you are using Vertica for SQL on Apache Hadoop, you typically place ROS data only on HDFS.

If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Requirements for HDFS Storage Locations

Caution:
If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

To store Vertica's data on HDFS, verify that:

- Your Hadoop cluster has WebHDFS enabled.
- All of the nodes in your Vertica cluster can connect to all of the nodes in your Hadoop cluster. Any firewall between the two clusters must allow connections on the ports used by HDFS.
- If your HDFS cluster is unsecured, you have a Hadoop user whose username matches the name of the Vertica database administrator (usually named dbadmin). This Hadoop user must have read and write access to the HDFS directory where you want Vertica to store its data.
- If your HDFS cluster uses Kerberos authentication, you have a Kerberos principal for Vertica, and it has read and write access to the HDFS directory that will be used for the storage location. See Configuring Kerberos. The Kerberos KDC must also be running.
- Your HDFS cluster has enough storage available for Vertica data. See HDFS Space Requirements below for details.

- The data you store in an HDFS-backed storage location does not expand your database's size beyond any data allowance in your Vertica license. Vertica counts data stored in an HDFS-backed storage location as part of any data allowance set by your license. See Managing Licenses in the Administrator's Guide for more information.

### HDFS Space Requirements

If your Vertica database is K-safe, HDFS-based storage locations contain two copies of the data you store in them. One copy is the primary projection, and the other is the buddy projection. If you have enabled HDFS's data-redundancy feature, Hadoop stores both projections multiple times. This duplication might seem excessive. However, it is similar to how a RAID level 1 or higher stores redundant copies of both the primary and buddy projections. The redundant copies also help the performance of HDFS by enabling multiple nodes to process a request for a file.

Verify that your HDFS installation has sufficient space available for redundant storage of both the primary and buddy projections of your K-safe data. You can adjust the number of duplicates stored by HDFS by setting the HadoopFSReplication configuration parameter. See Troubleshooting HDFS Storage Locations for details.

### Additional Requirements for Backing Up Data Stored on HDFS

To back up your data stored in HDFS storage locations, your Hadoop cluster must have snapshotting enabled for the directories to be used for backups. The easiest way to do this is to give the database administrator's account superuser privileges in Hadoop, so that snapshotting can be set automatically. Alternatively, use Hadoop to enable snapshotting for each directory before using it for backups.

In addition, your Vertica database must:

- Have enough Hadoop components and libraries installed to run the Hadoop distcp command as the Vertica database-administrator user (usually dbadmin).

- Have the JavaBinaryForUDx and HadoopHome configuration parameters set correctly.
Caution: After you have created an HDFS storage location, full database backups will fail with the error message:

ERROR 5127: Unable to create snapshot No such file /usr/bin/hadoop: check the HadoopHome configuration parameter

This error is caused by the backup script not being able to back up the HDFS storage locations. You must configure Vertica and Hadoop to enable the backup script to back these locations. After you configure Vertica and Hadoop, you can once again perform full database backups.

See Backing Up HDFS Storage Locations for details on configuring your Vertica and Hadoop clusters to enable HDFS storage location backup.

Best Practices for SQL on Apache Hadoop

If you are using the Vertica for SQL on Apache Hadoop product, Vertica recommends the following best practices for storage locations:

- Place only data type storage locations on HDFS storage.
- Place temp space directly on the local Linux file system, not in HDFS.
- For the best performance, place the Vertica catalog directly on the local Linux file system.
- Create the database first on a local Linux file system. Then, you can extend the database to HDFS storage locations and set storage policies that exclusively place data blocks on the HDFS storage location.
- For better performance, if you are running Vertica only on a subset of the HDFS nodes, do not run the HDFS balancer on them. The HDFS balancer can move data blocks farther away, causing Vertica to read non-local data during query execution. Queries run faster if they do not require network I/O.

Generally, HDFS requires approximately 2 GB of memory for each node in the cluster. To support this requirement in your Vertica configuration:

1. Create a 2-GB resource pool.
2. Do not assign any Vertica execution resources to this pool. This approach reserves the space for use by HDFS.

Alternatively, use Ambari or Cloudera Manager to find the maximum heap size required by HDFS and set the size of the resource pool to that value.
For more about how to configure resource pools, see Managing Workloads.

**How the HDFS Storage Location Stores Data**

Vertica stores data in storage locations on HDFS similarly to the way it stores data in the Linux file system. See Managing Storage Locations in the Administrator's Guide for more information about storage locations. When you create a storage location on HDFS, Vertica stores the ROS containers holding its data on HDFS. You can choose which data uses the HDFS storage location: from the data for just a single table or partition to all of the database's data.

When Vertica reads data from or writes data to an HDFS storage location, the node storing or retrieving the data contacts the Hadoop cluster directly to transfer the data. If a single ROS container file is split among several HDFS nodes, the Vertica node connects to each of them. The Vertica node retrieves the pieces and reassembles the file. Because each node fetches its own data directly from the source, data transfers are parallel, increasing their efficiency. Having the Vertica nodes directly retrieve the file splits also reduces the impact on the Hadoop cluster.

**What You Can Store in HDFS**

Use HDFS storage locations to store only data. You cannot store catalog information in an HDFS storage location.

Caution: While it is possible to use an HDFS storage location for temporary data storage, you must never do so. Using HDFS for temporary storage causes severe performance issues.

**What HDFS Storage Locations Cannot Do**

Because Vertica uses storage locations to store ROS containers in a proprietary format, MapReduce and other Hadoop components cannot access your Vertica ROS data stored in HDFS. Never allow another program that has access to HDFS to write to the ROS files. Any outside modification of these files can lead to data corruption and loss. Applications must use the Vertica client libraries to access Vertica data. If you want to share ROS data with other Hadoop components, you can export it (see Exporting Data).

The storage location stores and reads only ROS containers. It cannot read data stored in native formats in HDFS. See Hadoop Interfaces for other ways to read data stored in HDFS.
Creating an HDFS Storage Location

Use the CREATE LOCATION statement to create an HDFS storage location. Make the following changes from creating local storage locations:

- For the path, use the hdfs:// URL for the HDFS directory where you want Vertica to store the location's data. See Reading Directly from HDFS for information about the URL format.

- Include the ALL NODES SHARED keywords, as all HDFS storage locations are shared storage. This is required even if you have only one HDFS node in your cluster.

Caution:
If you use any HDFS storage locations, the HDFS data must be available at the time you start Vertica. Your HDFS cluster must be operational, and the ROS files must be present. If you have moved data files, or if they have become corrupted, or if your HDFS cluster is not responsive, Vertica cannot start.

Creating the Storage Location

To create an HDFS storage location, first create the location on all nodes and then set its storage policy to HDFS. To create the location in HDFS on all nodes:

```sql
=> CREATE LOCATION 'hdfs:///user/dbadmin' ALL NODES SHARED
   USAGE 'data' LABEL 'coldstorage';
```

Next, set the storage policy for your database objects to use this location:

```sql
=> SELECT SET_OBJECT_STORAGE_POLICY('SchemaName', 'coldstorage');
```

This causes all data in the named schema to be written to the HDFS storage location (coldstorage) instead of the local disk. You can set storage policies for a schema, a table, a partition, or the entire database.

For more information, see Managing Storage Locations.
Adding HDFS Storage Locations to New Nodes

If you add nodes to your Vertica cluster, they do not automatically have access to existing HDFS storage locations. You must manually create the storage location for the new node using the CREATE LOCATION statement. Do not use the ALL NODES keyword in this statement. Instead, use the NODE keyword with the name of the new node to tell Vertica that just that node needs to add the shared location.

Caution: You must manually create the storage location. Otherwise, the new node uses the default storage policy (usually, storage on the local Linux filesystem) to store data that the other the nodes store in HDFS. As a result, the node can run out of disk space.

The following example shows how to add the storage location from the preceding example to a new node named v_vmart_node0004:

```sql
=> CREATE LOCATION 'hdfs:///user/dbadmin' NODE 'v_vmart_node0004' 
   SHARED USAGE 'data' LABEL 'coldstorage';
```

Any active standby nodes in your cluster when you create an HDFS-based storage location automatically create their own instances of the location. When the standby node takes over for a down node, it uses its own instance of the location to store data for objects using the HDFS-based storage policy. Treat standby nodes added after you create the storage location as any other new node. You must manually define the HDFS storage location.

Creating a Storage Policy for HDFS Storage Locations

After you create an HDFS storage location, you assign database objects to the location by setting storage policies. Based on these storage policies, database objects such as partition ranges, individual tables, whole schemas, or even the entire database store their data in the HDFS storage location. Use the SET_OBJECT_STORAGE_POLICY function to assign objects to an HDFS storage location. In the function call, supply the label you assigned to the HDFS storage location as the location label argument. You do so using the CREATE LOCATION statement's LABEL keyword.

The following example demonstrates using SET_OBJECT_STORAGE_POLICY to store a table in an HDFS storage location. The example statement sets the policy for an existing table, named messages, to store its data in an HDFS storage location, named coldstorage.
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'coldstorage');

This table's data is moved to the HDFS storage location with the next merge-out. Alternatively, you can have Vertica move the data immediately by using the enforce_storage_move parameter.

You can query the STORAGE_CONTAINERS system table and examine the location_label column to verify that Vertica has moved the data:

```sql
=> SELECT node_name, projection_name, location_label, total_row_count FROM V_MONITOR.STORAGE_CONTAINERS
WHERE projection_name ILIKE 'messages%';
```

<table>
<thead>
<tr>
<th>node_name</th>
<th>projection_name</th>
<th>location_label</th>
<th>total_row_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_vmart_node0001</td>
<td>messages_b0</td>
<td>coldstorage</td>
<td>366057</td>
</tr>
<tr>
<td>v_vmart_node0001</td>
<td>messages_b1</td>
<td>coldstorage</td>
<td>366511</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>messages_b0</td>
<td>coldstorage</td>
<td>367432</td>
</tr>
<tr>
<td>v_vmart_node0002</td>
<td>messages_b1</td>
<td>coldstorage</td>
<td>366057</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>messages_b0</td>
<td>coldstorage</td>
<td>366511</td>
</tr>
<tr>
<td>v_vmart_node0003</td>
<td>messages_b1</td>
<td>coldstorage</td>
<td>367432</td>
</tr>
</tbody>
</table>

(6 rows)

See Creating Storage Policies in the Administrator's Guide for more information about assigning storage policies to objects.

## Backing Up HDFS Storage Locations

Vertica recommends that you regularly back up the data in your Vertica database. This recommendation includes data stored in your HDFS storage locations. The Vertica backup script (vbr) can back up HDFS storage locations. However, you must perform several configuration steps before it can back up these locations.

**Caution:** After you have created an HDFS storage location, full database backups will fail with the error message:

```
ERROR 5127: Unable to create snapshot No such file /usr/bin/hadoop: check the HadoopHome configuration parameter
```

This error is caused by the backup script not being able to back up the HDFS storage locations. You must configure Vertica and Hadoop to enable the backup script to back these locations. After you configure Vertica and Hadoop, you can once again perform full database backups.

There are several considerations for backing up HDFS storage locations in your database:
• HDFS storage locations do not support object-level backups. You must perform a full database backup to back up the data in your HDFS storage locations.

• Data in an HDFS storage location is backed up to HDFS. This backup guards against accidental deletion or corruption of data. It does not prevent data loss in the case of a catastrophic failure of the entire Hadoop cluster. To prevent data loss, you must have a backup and disaster recovery plan for your Hadoop cluster.

Data stored on the Linux native filesystem is still backed up to the location you specify in the backup configuration file. It and the data in HDFS storage locations are handled separately by the vbr backup script.

• You must configure your Vertica cluster to restore database backups containing an HDFS storage location. See Configuring Vertica to Restore HDFS Storage Locations for the configuration steps you must take.

• The HDFS directory for the storage location must have snapshotting enabled. You can either directly configure this yourself or enable the database administrator’s Hadoop account to do it for you automatically. See Configuring Hadoop and Vertica to Enable Backup of HDFS Storage for more information.

The topics in this section explain the configuration steps you must take to enable the backup of HDFS storage locations.

Configuring Hadoop and Vertica to Enable Backup of HDFS Storage

The Vertica backup script uses HDFS's snapshotting feature to create a backup of HDFS storage locations. A directory must allow snapshotting before HDFS can take a snapshot. Only a Hadoop superuser can enable snapshotting on a directory. Vertica can enable snapshotting automatically if the database administrator is also a Hadoop superuser.

If HDFS is unsecured, the following instructions apply to the database administrator account, usually dbadmin. If HDFS uses Kerberos security, the following instructions apply to the principal stored in the Vertica keytab file, usually vertica. The instructions below use the term "database account" to refer to this user.

We recommend that you make the database administrator or principal a Hadoop superuser. If you are not able to do so, you must enable snapshotting on the directory before configuring it for use by Vertica.
The steps you need to take to make the Vertica database administrator account a superuser depend on the distribution of Hadoop you are using. Consult your Hadoop distribution's documentation for details.

Manually Enabling Snapshotting for a Directory

If you cannot grant superuser status to the database account, you can instead enable snapshotting of each directory manually. Use the following command:

```
hdfs dfsadmin -allowSnapshot path
```

Issue this command for each directory on each node. Remember to do this each time you add a new node to your HDFS cluster.

Nested snapshottable directories are not allowed, so you cannot enable snapshotting for a parent directory to automatically enable it for child directories. You must enable it for each individual directory.

Additional Requirements for Kerberos

If HDFS uses Kerberos, then in addition to granting the keytab principal access, you must set a Vertica configuration parameter. In Vertica, set the HadoopConfDir parameter to the location of the directory containing the core-site.xml, hdfs-site.xml, and yarn-site.xml configuration files:

```
=> ALTER DATABASE exampledb SET HadoopConfDir = '/etc/hadoop/conf';
```

All three configuration files must be present in this directory.

If your Vertica nodes are not co-located on HDFS nodes, then you must copy these files from an HDFS node to each Vertica node. Use the same path on every database node, because HadoopConfDir is a global value.

Testing the Database Account's Ability to Make HDFS Directories Snapshottable

After making the database account a Hadoop superuser, verify that the account can set directories snapshottable:
1. Log into the Hadoop cluster as the database account (dbadmin by default).

2. Determine a location in HDFS where the database administrator can create a directory. The /tmp directory is usually available. Create a test HDFS directory using the command:

   ```
   $ hdfs dfs -mkdir /path/testdir
   ```

3. Make the test directory snapshottable using the command:

   ```
   $ hdfs dfsadmin -allowSnapshot /path/testdir
   ```

The following example demonstrates creating an HDFS directory and making it snapshottable:

```
$ hdfs dfs -mkdir /tmp/snaptest
$ hdfs dfsadmin -allowSnapshot /tmp/snaptest
Allowing snapshot on /tmp/snaptest succeeded
```

### Configuring Vertica to Restore HDFS Storage Locations

Your Vertica cluster must be able to run the Hadoop distcp command to restore a backup of an HDFS storage location. The easiest way to enable your cluster to run this command is to install several Hadoop packages on each node. These packages must be from the same distribution and version of Hadoop that is running on your Hadoop cluster.

The steps you need to take depend on:

- The distribution and version of Hadoop running on the Hadoop cluster containing your HDFS storage location.
- The distribution of Linux running on your Vertica cluster.

**Note:** Installing the Hadoop packages necessary to run distcp does not turn your Vertica database into a Hadoop cluster. This process installs just enough of the Hadoop support files on your cluster to run the distcp command. There is no additional overhead placed on the Vertica cluster, aside from a small amount of additional disk space consumed by the Hadoop support files.

### Configuration Overview

The steps for configuring your Vertica cluster to restore backups for HDFS storage location are:
1. If necessary, install and configure a Java runtime on the hosts in the Vertica cluster.

2. Find the location of your Hadoop distribution's package repository.

3. Add the Hadoop distribution's package repository to the Linux package manager on all hosts in your cluster.

4. Install the necessary Hadoop packages on your Vertica hosts.

5. Set two configuration parameters in your Vertica database related to Java and Hadoop.

6. If your HDFS storage location uses Kerberos, set additional configuration parameters to allow Vertica user credentials to be proxied.

7. Confirm that the Hadoop distcp command runs on your Vertica hosts.

The following sections describe these steps in greater detail.

## Installing a Java Runtime

You Vertica cluster must have a Java Virtual Machine (JVM) installed to run the Hadoop distcp command. It already has a JVM installed if you have configured it to:

- Execute User-Defined Extensions developed in Java. See [Developing User-Defined Extensions (UDxs)](https://docs.vertica.com/9.0.x/en-us/udx.html) for more information.

- Access Hadoop data using the HCatalog Connector. See [Using the HCatalog Connector](https://docs.vertica.com/9.0.x/en-us/hcatalogconnector.html) for more information.

If your Vertica database has a JVM installed, verify that your Hadoop distribution supports it. See your Hadoop distribution's documentation to determine which JVMs it supports.

If the JVM installed on your Vertica cluster is not supported by your Hadoop distribution you must uninstall it. Then you must install a JVM that is supported by both Vertica and your Hadoop distribution. See [Vertica SDKs in Supported Platforms](https://docs.vertica.com/9.0.x/en-us/vertica-platforms.html) for a list of the JVMs compatible with Vertica.

If your Vertica cluster does not have a JVM (or its existing JVM is incompatible with your Hadoop distribution), follow the instructions in [Installing the Java Runtime on Your Vertica Cluster](https://docs.vertica.com/9.0.x/en-us/installing/java-runtime.html).
Finding Your Hadoop Distribution's Package Repository

Many Hadoop distributions have their own installation system, such as Cloudera's Manager or Hortonwork's Ambari. However, they also support manual installation using native Linux packages such as RPM and .deb files. These package files are maintained in a repository. You can configure your Vertica hosts to access this repository to download and install Hadoop packages.

Consult your Hadoop distribution's documentation to find the location of its Linux package repository. This information is often located in the portion of the documentation covering manual installation techniques.

Each Hadoop distribution maintains separate repositories for each of the major Linux package management systems. Find the specific repository for the Linux distribution running on your Vertica cluster. Be sure that the package repository that you select matches the version of Hadoop distribution installed on your Hadoop cluster.

Configuring Vertica Nodes to Access the Hadoop Distribution’s Package Repository

Configure the nodes in your Vertica cluster so they can access your Hadoop distribution's package repository. Your Hadoop distribution's documentation should explain how to add the repositories to your Linux platform. If the documentation does not explain how to add the repository to your packaging system, refer to your Linux distribution's documentation.

The steps you need to take depend on the package management system your Linux platform uses. Usually, the process involves:

- Downloading a configuration file.
- Adding the configuration file to the package management system's configuration directory.
- For Debian-based Linux distributions, adding the Hadoop repository encryption key to the root account keyring.
- Updating the package management system's index to have it discover new packages.

The following example demonstrates adding the Hortonworks 2.1 package repository to an Ubuntu 12.04 host. These steps in this example are explained in the Hortonworks documentation.
You must add the Hadoop repository to all hosts in your Vertica cluster.

### Installing the Required Hadoop Packages

After configuring the repository, you are ready to install the Hadoop packages. The packages you need to install are:

- `hadoop`
- `hadoop-hdfs`
hadoop-client

The names of the packages are usually the same across all Hadoop and Linux distributions. These packages often have additional dependencies. Always accept any additional packages that the Linux package manager asks to install.

To install these packages, use the package manager command for your Linux distribution. The package manager command you need to use depends on your Linux distribution:

- On Red Hat and CentOS, the package manager command is `yum`
- On Debian and Ubuntu, the package manager command is `apt-get`
- On SUSE, the package manager command is `zypper`

Consult your Linux distribution's documentation for instructions on installing packages.

The following example demonstrates installing the required Hadoop packages from the Hortonworks 2.1 distribution on an Ubuntu 12.04 system.

```bash
# apt-get install hadoop hadoop-hdfs hadoop-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bigtop-jsvc hadoop-mapreduce hadoop-yarn zookeeper
The following NEW packages will be installed:
  bigtop-jsvc hadoop hadoop-client hadoop-hdfs hadoop-mapreduce hadoop-yarn
Zookeeper
0 upgraded, 7 newly installed, 0 to remove and 90 not upgraded.
Need to get 86.6 MB of archives.
After this operation, 99.8 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  bigtop-jsvc amd64 1.8.10-1 [28.5 kB]
Get:2 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  zookeeper all 3.4.5.2.1.3.0-563 [6,828 kB]
Get:3 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop all 2.4.0.2.1.3.0-563 [21.5 MB]
Get:4 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-hdfs all 2.4.0.2.1.3.0-563 [16.0 MB]
Get:5 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-yarn all 2.4.0.2.1.3.0-563 [15.1 MB]
Get:6 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-mapreduce all 2.4.0.2.1.3.0-563 [27.2 MB]
Get:7 http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.1.3.0/ HDP/main
  hadoop-client all 2.4.0.2.1.3.0-563 [3,650 B]
Fetched 86.6 MB in 1min 2s (1,396 kB/s)
Selecting previously unselected package bigtop-jsvc.
(Reading database ... 197894 files and directories currently installed.)
Unpacking bigtop-jsvc (from .../bigtop-jsvc_1.8.10-1_amd64.deb) ...
Selecting previously unselected package zookeeper.
Unpacking zookeeper (from .../zookeeper_3.4.5.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop.
```
Unpacking hadoop (from .../hadoop_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-hdfs.
Unpacking hadoop-hdfs (from .../hadoop-hdfs_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-yarn.
Unpacking hadoop-yarn (from .../hadoop-yarn_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-mapreduce.
Unpacking hadoop-mapreduce (from .../hadoop-mapreduce_2.4.0.2.1.3.0-563_all.deb) ...
Selecting previously unselected package hadoop-client.
Unpacking hadoop-client (from .../hadoop-client_2.4.0.2.1.3.0-563_all.deb) ...
Processing triggers for man-db ...
Setting up bigtop-jsvc (1.0.10-1) ...
Setting up zookeeper (3.4.5.2.1.3.0-563) ...
update-alternatives: using /etc/zookeeper/conf.dist to provide /etc/zookeeper/conf (zookeeper-conf)
in auto mode.
Setting up hadoop (2.4.0.2.1.3.0-563) ...
update-alternatives: using /etc/hadoop/conf.empty to provide /etc/hadoop/conf (hadoop-conf) in auto mode.
Setting up hadoop-hdfs (2.4.0.2.1.3.0-563) ...
Setting up hadoop-yarn (2.4.0.2.1.3.0-563) ...
Setting up hadoop-mapreduce (2.4.0.2.1.3.0-563) ...
Setting up hadoop-client (2.4.0.2.1.3.0-563) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place

Setting Configuration Parameters

You must set two configuration parameters to enable Vertica to restore HDFS data:

- **JavaBinaryForUDx** is the path to the Java executable. You may have already set this value to use Java UDxs or the HCatalog Connector. You can find the path for the default Java executable from the Bash command shell using the command:

  ```
  which java
  ```

- **HadoopHome** is the path where Hadoop is installed on the Vertica hosts. This is the directory that contains bin/hadoop (the bin directory containing the Hadoop executable file). The default value for this parameter is /usr. The default value is correct if your Hadoop executable is located at /usr/bin/hadoop.

The following example demonstrates setting and then reviewing the values of these parameters.

```sql
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';
=> SELECT get_config_parameter('JavaBinaryForUDx');
get_config_parameter
----------------------
/usr/bin/java
(1 row)
=> ALTER DATABASE mydb SET HadoopHome = '/usr';
```
There are additional parameters you may, optionally, set:

- **HadoopFSReadRetryTimeout** and **HadoopFSWriteRetryTimeout** specify how long to wait before failing. The default value for each is 180 seconds. If you are confident that your file system will fail more quickly, you can potentially improve performance by lowering these values.

- **HadoopFSReplication** is the number of replicas HDFS makes. By default the Hadoop client chooses this; Vertica uses the same value for all nodes. We recommend against changing this unless directed to.

- **HadoopFSBlockSizeBytes** is the block size to write to HDFS; larger files are divided into blocks of this size. The default is 64MB.

### Setting Kerberos Parameters

If your Vertica nodes are co-located on HDFS nodes and you are using Kerberos, you must change some Hadoop configuration parameters. These changes are needed in order for restoring from backups to work. In yarn-site.xml on every Vertica node, set the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.proxy-user-privileges.enabled</td>
<td>true</td>
</tr>
<tr>
<td>yarn.resourcemanager.proxyusers.*.groups</td>
<td>*</td>
</tr>
<tr>
<td>yarn.resourcemanager.proxyusers.*.hosts</td>
<td>*</td>
</tr>
<tr>
<td>yarn.resourcemanager.proxyusers.*.users</td>
<td>*</td>
</tr>
<tr>
<td>yarn.timeline-service.http-authentication.proxyusers.*.groups</td>
<td>*</td>
</tr>
<tr>
<td>yarn.timeline-service.http-authentication.proxyusers.*.hosts</td>
<td>*</td>
</tr>
<tr>
<td>yarn.timeline-service.http-authentication.proxyusers.*.users</td>
<td>*</td>
</tr>
</tbody>
</table>

No changes are needed on HDFS nodes that are not also Vertica nodes.
Confirming that distcp Runs

After the packages are installed on all hosts in your cluster, your database should be able to run the Hadoop distcp command. To test it:

1. Log into any host in your cluster as the database administrator.
2. At the Bash shell, enter the command:

   ```
   $ hadoop distcp
   ```

3. The command should print a message similar to the following:

   ```
   usage: distcp OPTIONS [source_path...] <target_path>
   OPTIONS
   -async Should distcp execution be blocking
   -atomic Commit all changes or none
   -bandwidth <arg> Specify bandwidth per map in MB
   -delete Delete from target, files missing in source
   -f <arg> List of files that need to be copied
   -filelimit <arg> (Deprecated!) Limit number of files copied to <= n
   -i Ignore failures during copy
   -log <arg> Folder on DFS where distcp execution logs are saved
   -m <arg> Max number of concurrent maps to use for copy
   -mapredSslConf <arg> Configuration for ssl config file, to use with https://
   -overwrite Choose to overwrite target files unconditionally, even if they exist.
   -p <arg> preserve status (rbugpc)(replication, block-size, user, group, permission, checksum-type)
   -sizelimit <arg> (Deprecated!) Limit number of files copied to <= n bytes
   -skipcrccheck Whether to skip CRC checks between source and target paths.
   -strategy <arg> Copy strategy to use. Default is dividing work based on file sizes
   -tmp <arg> Intermediate work path to be used for atomic commit
   -update Update target, copying only missing files or directories
   ```

4. Repeat these steps on the other hosts in your database to verify that all of the hosts can run distcp.

Troubleshooting

If you cannot run the distcp command, try the following steps:
If Bash cannot find the hadoop command, you may need to manually add Hadoop's bin directory to the system search path. An alternative is to create a symbolic link in an existing directory in the search path (such as /usr/bin) to the hadoop binary.

Ensure the version of Java installed on your Vertica cluster is compatible with your Hadoop distribution.

Review the Linux package installation tool's logs for errors. In some cases, packages may not be fully installed, or may not have been downloaded due to network issues.

Ensure that the database administrator account has permission to execute the hadoop command. You may need to add the account to a specific group in order to allow it to run the necessary commands.

Performing Backups Containing HDFS Storage Locations

After you configure Hadoop and Vertica, HDFS storage locations are automatically backed up when you perform a full database backup. If you already have a backup configuration file for a full database backup, you do not need to make any changes to it. You just run the vbr backup script as usual to perform the full database backup. See Creating Full Backups in the Administrator's Guide for instructions on running the vbr backup script.

If you do not have a backup configuration file for a full database backup, you must create one to back up the data in your HDFS storage locations. See Creating vbr Configuration Files in the Administrator's Guide for more information.

Removing HDFS Storage Locations

The steps to remove an HDFS storage location are similar to standard storage locations:

1. Remove any existing data from the HDFS storage location by using SET_OBJECT_STORAGE_POLICY to change each object's storage location. Alternatively, you can use CLEAR_OBJECT_STORAGE_POLICY. Because the Tuple Mover runs infrequently, set the enforce_storage_move parameter to true to make the change immediately.

2. Retire the location on each host that has the storage location defined by using RETIRE_LOCATION. Set enforce_storage_move to true.
3. Drop the location on each host that has the storage location defined by using `DROP_LOCATION`.

4. Optionally remove the snapshots and files from the HDFS directory for the storage location.

For more information about changing storage policies, changing usage, retiring locations, and dropping locations, see Managing Storage Locations in the Administrator's Guide.

Important: If you have backed up the data in the HDFS storage location you are removing, you must perform a full database backup after you remove the location. If you do not and restore the database to a backup made before you removed the location, the location's data is restored.

Removing Storage Location Files from HDFS

Dropping an HDFS storage location does not automatically clean the HDFS directory that stored the location's files. Any snapshots of the data files created when backing up the location are also not deleted. These files consume disk space on HDFS and also prevent the directory from being reused as an HDFS storage location. Vertica refuses to create a storage location in a directory that contains existing files or subdirectories. You must log into the Hadoop cluster to delete the files from HDFS. An alternative is to use some other HDFS file management tool.

Removing Backup Snapshots

HDFS returns an error if you attempt to remove a directory that has snapshots:

```bash
$ hdfs dfs -rm -r -f -skipTrash /user/dbadmin/v_vmart_node0001
rm: The directory /user/dbadmin/v_vmart_node0001 cannot be deleted since
/user/dbadmin/v_vmart_node0001 is snapshottable and already has snapshots
```

The Vertica backup script creates snapshots of HDFS storage locations as part of the backup process. See Backing Up HDFS Storage Locations for more information. If you made backups of your HDFS storage location, you must delete the snapshots before removing the directories.

HDFS stores snapshots in a subdirectory named `.snapshot`. You list the snapshots in the directory using the standard HDFS `ls` command. The following example demonstrates listing the snapshots defined for node0001.

```bash
$ hdfs dfs -ls /user/dbadmin/v_vmart_node0001/.snapshot
Found 1 items
```
To remove snapshots, use the command:

```bash
hdfs dfs -removeSnapshot directory snapshotname
```

The following example demonstrates the command to delete the snapshot shown in the previous example:

```bash
$ hdfs dfs -deleteSnapshot /user/dbadmin/v_vmart_node0001 s20140902-101358.629
```

You must delete each snapshot from the directory for each host in the cluster. After you have deleted the snapshots, you can delete the directories in the storage location.

**Important:** Each snapshot's name is based on a timestamp down to the millisecond. Nodes independently create their own snapshot. They do not synchronize snapshot creation, so their snapshot names differ. You must list each node's snapshot directory to learn the names of the snapshots it contains.

See Apache's [HDFS Snapshot documentation](https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-dfs/HdfsDesign.html) for more information about managing and removing snapshots.

### Removing the Storage Location Directories

You can remove the directories that held the storage location's data by either of the following methods:

- Use an HDFS file manager to delete directories. See your Hadoop distribution's documentation to determine if it provides a file manager.
- Log into the Hadoop NameNode using the database administrator’s account and use HDFS's `rmr` command to delete the directories. See Apache's [File System Shell Guide](https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-dfs/DFSCommandLineInterface.html) for more information.

The following example uses the HDFS `rmr` command from the Linux command line to delete the directories left behind in the HDFS storage location directory `/user/dbadmin`. It uses the `-skipTrash` flag to force the immediate deletion of the files.

```bash
$ hdfs dfs -ls /user/dbadmin
 Found 3 items
 drwxrwx--x  dbadmin supergroup  0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0001
 drwxrwx--x  dbadmin supergroup  0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0002
 drwxrwx--x  dbadmin supergroup  0 2014-08-29 15:11 /user/dbadmin/v_vmart_node0003

$ hdfs dfs -rmr -skipTrash /user/dbadmin/*
```
Troubleshooting HDFS Storage Locations

This topic explains some common issues with HDFS storage locations.

HDFS Storage Disk Consumption

By default, HDFS makes three copies of each file it stores. This replication helps prevent data loss due to disk or system failure. It also helps increase performance by allowing several nodes to handle a request for a file.

A Vertica database with a K-Safety value of 1 or greater also stores its data redundantly using buddy projections.

When a K-Safe Vertica database stores data in an HDFS storage location, its data redundancy is compounded by HDFS's redundancy. HDFS stores three copies of the primary projection's data, plus three copies of the buddy projection for a total of six copies of the data.

If you want to reduce the amount of disk storage used by HDFS locations, you can alter the number of copies of data that HDFS stores. The Vertica configuration parameter named HadoopFSReplication controls the number of copies of data HDFS stores.

You can determine the current HDFS disk usage by logging into the Hadoop NameNode and issuing the command:

```bash
hdfs dfsadmin -report
```

This command prints the usage for the entire HDFS storage, followed by details for each node in the Hadoop cluster. The following example shows the beginning of the output from this command, with the total disk space highlighted:

```
$ hdfs dfsadmin -report
Configured Capacity: 51495516981 (47.96 GB)
Present Capacity: 32087212032 (29.88 GB)
DFS Remaining: 31565144064 (29.40 GB)
DFS Used: 522067968 (497.88 MB)
DFS Used%: 1.63%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
...
After loading a simple million-row table into a table stored in an HDFS storage location, the report shows greater disk usage:

```
Configured Capacity: 51495516981 (47.96 GB)
Present Capacity: 32085299338 (29.88 GB)
DFS Remaining: 31373565952 (29.22 GB)
DFS Used: 71173386 (678.76 MB)
DFS Used%: 2.22%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
...```

The following Vertica example demonstrates:

1. Dropping the table in Vertica.

2. Setting the HadoopFSReplication configuration option to 1. This tells HDFS to store a single copy of an HDFS storage location’s data.

3. Recreating the table and reloading its data.

```
=> DROP TABLE messages;
DROP TABLE
=> ALTER DATABASE mydb SET HadoopFSReplication = 1;
=> CREATE TABLE messages (id INTEGER, text VARCHAR);
CREATE TABLE
=> SELECT SET_OBJECT_STORAGE_POLICY('messages', 'hdfs');
SET_OBJECT_STORAGE_POLICY
-------------------------
Object storage policy set.
(1 row)
=> COPY messages FROM '/home/dbadmin/messages.txt' DIRECT;
Rows Loaded
-----------
1000000
```

Running the HDFS report on Hadoop now shows less disk space use:

```
$ hdfs dfsadmin -report
Configured Capacity: 51495516981 (47.96 GB)
Present Capacity: 32086278190 (29.88 GB)
DFS Remaining: 31500988416 (29.34 GB)
DFS Used: 585289774 (558.18 MB)
DFS Used%: 1.82%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
...```
Caution: Reducing the number of copies of data stored by HDFS increases the risk of data loss. It can also negatively impact the performance of HDFS by reducing the number of nodes that can provide access to a file. This slower performance can impact the performance of Vertica queries that involve data stored in an HDFS storage location.

Kerberos Authentication When Creating a Storage Location

If HDFS uses Kerberos authentication, then the CREATE LOCATION statement authenticates using the Vertica keytab principal, not the principal of the user performing the action. If the creation fails with an authentication error, verify that you have followed the steps described in Configuring Kerberos to configure this principal.

When creating an HDFS storage location on a Hadoop cluster using Kerberos, CREATE LOCATION reports the principal being used as in the following example:

```
=> CREATE LOCATION 'hdfs:///user/dbadmin' ALL NODES SHARED
    USAGE 'data' LABEL 'coldstorage';
NOTICE 0: Performing HDFS operations using kerberos principal [vertica/hadoop.example.com]
CREATE LOCATION
```

Backup or Restore Fails When Using Kerberos

When backing up an HDFS storage location that uses Kerberos, you might see an error such as:

```
createSnapshot: Failed on local exception: java.io.IOException:
java.lang.IllegalArgumentException: Server has invalid Kerberos principal:
hdfs/test.example.com@EXAMPLE.COM;
```

When restoring an HDFS storage location that uses Kerberos, you might see an error such as:

```
Error msg: Initialization thread logged exception: 
   Distcp failure!
```

Either of these failures means that Vertica could not find the required configuration files in the HadoopConfDir directory. Usually this is because you have set the parameter but not copied the files from an HDFS node to your Vertica node. See "Additional Requirements for Kerberos" in Configuring Hadoop and Vertica to Enable Backup of HDFS Storage.
Integrating With Cloudera Manager

The Cloudera distribution of Hadoop includes Cloudera Manager, a web-based tool for managing a Hadoop cluster. Cloudera Manager can manage any service for which a service description is available, including Vertica.

You can use Cloudera Manager to start, stop, and monitor individual database nodes or the entire database. You can manage both co-located and separate Vertica clusters—Cloudera can manage services on nodes that are not part of the Hadoop cluster.

You must install and configure your Vertica database before proceeding; you cannot use Cloudera Manager to create the database.

Installing the Service

Note: Because the service has to send the database password over the network, you should enable encryption on your Hadoop cluster before proceeding.

A Cloudera Service Description (CSD) file describes a service that Cloudera can manage. The Vertica CSD is in /opt/vertica/share/CSD on a database node.

To install the Vertica CSD, follow these steps:

1. On a Vertica node, follow the instructions in VerticaAPIKey to generate an API key. You need this key to finish the installation of the CSD.

2. On the Hadoop node that hosts Cloudera Manager, copy the CSD file into /opt/cloudera/csd.

3. Restart Cloudera Manager:

   
   ```
   $ service cloudera-scm-server restart
   ```

4. In a web browser, go to Cloudera Manager and restart the Cloudera Management Service.

5. If your Vertica cluster is separate from your Hadoop cluster (not co-located on it): Use Cloudera Manager to add the hosts for your database nodes. If your cluster is co-located, skip this step.

6. Use Cloudera Manager to add the Vertica service.
7. On the "Role Assignment" page, select the hosts that are database nodes.

8. On the "Configuration" page, specify values for the following fields:
   - database name
   - agent port (accept the default if you're not sure)
   - API key
   - database user to run as (usually dbadmin) and password

**About the Agent**

When you manage Vertica through Cloudera Manager, you are actually interacting with the Vertica Agent, not the database directly. The Agent runs on all database nodes and interacts with the database on your behalf. Management Console uses the same agent. Most of the time this extra indirection is transparent to you.

A Cloudera-managed service contains one or more roles. In this case the service is "Vertica" and the single role is "Vertica Node".

**Available Operations**

Cloudera Manager shows two groups of operations. Service-level operations apply to the service on all nodes, while role-level operations apply only to a single node.

You can perform the following service-level operations on all nodes:

- **Start**: Starts the agent and, if it is not already running, the database.
- **Stop**: Stops the database and agent.
- **Restart**: Calls Stop and then Start.
- **Add Role Instances**: Adds new database nodes to Cloudera Manager. The nodes must already be part of the Vertica cluster, and the hosts must already be known to Cloudera Manager.
- **Enter Maintenance Mode**: Suppresses health alerts generated by Cloudera Manager.
- **Exit Maintenance Mode**: Resumes normal reporting.
- Update Memory Pool Size: Applies memory-pool settings from the Static Service Pools configuration page.

You can perform all of these operations except Add Role Instances on individual nodes as role-level operations.

## Managing Memory Pools

Cloudera Manager allows you to change resource allocations, such as memory and CPU, for the nodes it manages. If you are using co-located clusters, centrally managing resources can simplify your cluster management. If you are using separate Hadoop and Vertica clusters, you might prefer to manage Vertica separately as described in Managing the Database in the Administrator's Guide.

Use the Cloudera Manager "Static Service Pools" configuration page to configure resource allocations. The "Vertica Memory Pool" value, specified in GB, is the maximum amount of memory to allocate to the database on each node. If the configuration page includes "Cgroup Memory Hard Limit", set it to the same value as "Vertica Memory Pool".

After you have set these values, you can use the "Update Memory Pool Size" operation to apply the value to the managed nodes. This operation is equivalent to ALTER RESOURCE POOL GENERAL MAXMEMORYSIZE. Configuration changes in "Static Service Pools" do not take effect in Vertica until you perform this operation.

## Uninstalling the Service

To uninstall the Vertica CSD, follow these steps:

1. Stop the Vertica service and then remove it from Cloudera Manager.
2. Remove the CSD file from /opt/cloudera/csd.
3. From the command line, restart the Cloudera Manager server.
4. In Cloudera Manager, restart the Cloudera Management Service.
Integrating Vertica with the MapR Distribution of Hadoop

MapR is a distribution of Apache Hadoop produced by MapR Technologies that extends the standard Hadoop components with its own features. Vertica can integrate with MapR in the following ways:

- You can read data from MapR through an NFS mount point. After you mount the MapR file system as an NFS mount point, you can use `CREATE EXTERNAL TABLE AS COPY` or `COPY` to access the data as if it were on the local file system. For ORC or Parquet data, see also Reading Hadoop Columnar File Formats. This option provides the best performance for reading data.

- You can use the HCatalog Connector to read Hive data. Do not use the HCatalog Connector with ORC or Parquet data in MapR for performance reasons. Instead, mount the MapR file system as an NFS mount point and create external tables without using the Hive schema. See Using the HCatalog Connector.

- You can create a storage location to store data in MapR using the native Vertica format (ROS). Mount the MapR file system as an NFS mount point and then use `CREATE LOCATION...ALL NODES SHARED` to create a storage location. (CREATE LOCATION does not support NFS mount points in general, but does support them for MapR.)

Other Vertica integrations for Hadoop are not available for MapR.

For information on mounting the MapR file system as an NFS mount point, see Accessing Data with NFS and Configuring Vertica Analytics Platform with MapR on the MapR website. In particular, you must configure MapR to add Vertica as a MapR service.

Examples

In the following examples, the MapR file system has been mounted as /mapr.

The following statement creates an external table from ORC data:

```sql
=> CREATE EXTERNAL TABLE t (a1 INT, a2 VARCHAR(20))
    AS COPY FROM '/mapr/data/file.orc' ORC;
```

The following statement creates an external table from Parquet data and takes advantage of partition pruning (see Using Partition Columns):
CREATE EXTERNAL TABLE t2 (id int, name varchar(50), created date, region varchar(50))
AS COPY FROM '/mapr/*/*/*' PARQUET(hive_partition_cols='created,region');

The following statement loads ORC data from MapR into Vertica:

COPY t FROM '/mapr/data/*.orc' ON ANY NODE ORC;

The following statement creates a storage location to hold ROS data in the MapR file system:

CREATE LOCATION '/mapr/my.cluster.com/data' SHARED USAGE 'DATA' LABEL 'maprfs';
Integrating with Apache Kafka

Welcome to the Vertica Data Streaming Integration Guide.
Audience

This book is intended for anyone who wants to load data from an existing data streaming message bus into a Vertica database.
Prerequisites

This document assumes that you have installed and configured Vertica as described in Installing Vertica and the Configuring the Database section of the Administrator's Guide. In addition, you must have Java 7.0 or later installed on your Vertica node. Refer to the Vertica product documentation to learn more.

In addition, this guide assumes you have installed and configured your data streaming platform. For details on installing and using third party applications, please refer to the documentation for that application.
How Vertica and Data Streaming Work Together

Vertica provides a high-performance mechanism for streaming data both to and from third party message buses. Data streaming provides high volumes of data with low latency. This feature lets you:

- Store and analyze data that is generated by any application that writes to the message bus. With a message bus, you do not need to worry about individually configuring each application to connect to Vertica.

- Send Vertica data to any application that can read from the message bus. There are two types of data you can send to a data bus:
  - Vertica tables and results of queries. This feature lets you export the results of Vertica analytics to any other application connected to the message bus.
  - Data from Vertica's Data Collector tables. You can use this data to monitor your Vertica database's performance and health via third-party monitoring tools.

Because Vertica can both receive and send data to a streaming message bus, you can use it as part of an automated analytics workflow. Vertica can retrieve data from the message bus, perform analytics on it, and then send the results back to the message bus for consumption by other applications.

Common Uses of Data Streaming with Vertica

There are many cases where you could use Vertica to process data from a streaming data source. This document will use the following two cases as examples:

- Retrieving and processing data from an e-commerce site. This includes raw access logs from a web server farm, clickstream data generated by a JavaScript-based Web 2.0 interface, and even back-end fulfillment processing.

- An Internet of Things (IOT) network where device sensor data is sent from many individual devices to the message bus via web API calls.
Data Streaming Integration Terms

Vertica uses the following terms to describe its streaming feature. These are general terms, which may differ from each specific streaming platform's terminology.

Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>A data streaming server.</td>
</tr>
<tr>
<td>Source</td>
<td>A feed of messages in a common category which streams into the same Vertica target tables. In Apache Kafka, a source is known as a topic.</td>
</tr>
<tr>
<td>Partition</td>
<td>Unit of parallelism within data streaming. Data streaming splits a source into multiple partitions, which can each be served in parallel to consumers such as a Vertica database. Within a partition, messages are usually ordered chronologically.</td>
</tr>
<tr>
<td>Offset</td>
<td>An index into a partition. This index is the position within an ordered queue of messages, not an index into an opaque byte stream.</td>
</tr>
<tr>
<td>Message</td>
<td>A unit of data within data streaming. The data is typically in JSON or Avro format. Messages are loaded as rows into Vertica tables, and are uniquely identified by their source, partition, and offset.</td>
</tr>
</tbody>
</table>

Data Loader Terminology

<table>
<thead>
<tr>
<th>Data Loader Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduler</td>
<td>An external tool that schedules data loads from a streaming data source into Vertica.</td>
</tr>
<tr>
<td>Microbatch</td>
<td>A microbatch represents a single segment of a data load from a</td>
</tr>
<tr>
<td>Data Loader Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>streaming data source. It encompasses all of the information the scheduler needs to perform a load from a streaming data source into Vertica.</td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>The window of time during which a Scheduler executes microbatches to load data. This window controls the duration of each COPY statement the scheduler runs as a part of the microbatch. The during the frame, the scheduler gives an active microbatch from each source an opportunity to load data. It gives priority to microbatches that need more time to load data based on the history of previous microbatches.</td>
</tr>
<tr>
<td>Stream</td>
<td>A feed of messages that is identified by a source and partition. The offset uniquely identifies the position within a particular source-partition stream.</td>
</tr>
<tr>
<td>Lane</td>
<td>A thread within a job scheduler instance that issues microbatches to perform the load. The number of lanes available is based on the PlannedConcurrency of the job scheduler's resource pool. Multiple lanes allow the scheduler to run microbatches for different sources in parallel during a frame.</td>
</tr>
</tbody>
</table>
Vertica and Apache Kafka

Currently, the only data streaming platform that Vertica supports is Apache Kafka. Kafka is an open-source distributed real-time streaming platform. See Apache's main Kafka page for more information. By integrating Kafka and Vertica, you can load data from any application that produces Kafka messages.

The integration features between Vertica and Kafka consist of:

- A UDx library containing functions that load and parse data from Kafka topics into Vertica.
- A job scheduler that uses the UDL library to continuously consume data from your message bus with exactly-once semantics
- Push-based Monitoring Vertica Using Notifiers that send data collector messages from Vertica to Kafka
- A KafkaExport function that sends Vertica data to Kafka.

Consuming Data from Kafka

There are two ways to load data from Kafka:
- Manually, by directly executing a COPY statement. You use this method to load a finite amount of data. Some reasons you may want to manually load data:
  
  - Managing a streaming data load with greater control than using a scheduler. Schedulers can manage data loads in many cases. However, you may find you need greater control over the data load than is available through the scheduler.
  
  - Loading specific chunks of data you want to analyze, rather than constantly streaming data. For example, you may want to load web server logs from a specific time period to perform in-depth analytics.
  
  - Manually testing your Kafka and Vertica configuration before creating a streaming data load. See Manually Copying Data From Kafka for more information.
  
- Automatically via job schedulers. The schedulers constantly load data from Kafka and ensure each Kafka message is loaded exactly once. See Automatically Copying Data From Kafka for more information.

### Producing Data for Kafka

Vertica can send data to Kafka for processing by other consumers of Kafka's data streams. There are two ways you can send data to Kafka:

- You use notifiers to send Vertica health and performance data stored in the Data Collector tables. This feature is useful to stream data to third-party monitoring tools.
  
- You use the KafkaExport function to export Vertica data to Kafka. See Producing Data Using KafkaExport for more information.
Kafka and Vertica Configuration Settings

The following sections lists settings for Vertica and Kafka that you can set to optimize performance.

Vertica Producer Settings

These settings change how Vertica acts as a Kafka producer when using either the KafkaExport function or through notifiers. How you change these settings depends on the method you are using to export data (KafkaExport or Notifiers).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue.buffering.max.messages</td>
<td>Specifies the size of the Vertica producer queue. If Vertica generates too many messages too quickly, the queue can fill, resulting in dropped messages. Increasing this value consumes more memory, but reduces the chance of lost messages.</td>
</tr>
<tr>
<td>queue.buffering.max.ms</td>
<td>Specifies the frequency with which Vertica flushes the producer message queue. Lower values decrease latency at the cost of throughput. Higher values increase throughput, but can cause the producer queue (set by queue.buffering.max.messages) to fill more frequently, resulting in dropped messages.</td>
</tr>
<tr>
<td>message.max.bytes</td>
<td>Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.</td>
</tr>
<tr>
<td>message.send.max.retries</td>
<td>Specifies the number of attempts the producer makes to deliver the message to a broker. Higher values increase the chance of success.</td>
</tr>
</tbody>
</table>
| retry.backoff.ms              | Specifies the interval Vertica waits before resending a
Setting | Notes
--- | ---
failed message. | 
request.required.acks | Specifies how many broker replica acknowledgments Kafka requires before it considers message delivery successful. Requiring acknowledgments increases latency. Removing acknowledgments increases the risk of message loss.
request.timeout.ms | Specifies the interval that the producer waits for a response from the broker. Broker response time is affected by server load and the number of message acknowledgments you require. Higher values increase latency.

**Kafka Broker Settings**

Kafka brokers receive messages from producers and distribute them among Kafka consumers. Configure these settings on the brokers themselves. These settings function independently of your producer and consumer settings. For detailed information on Apache Kafka broker settings, refer to the Apache Kafka documentation.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>message.max.bytes</td>
<td>Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.</td>
</tr>
<tr>
<td>num.io.threads</td>
<td>Specifies the number of network threads the broker uses to receive and process requests. More threads can increase your concurrency.</td>
</tr>
<tr>
<td>num.network.threads</td>
<td>Specifies the number of network threads the broker uses to accept network requests. More threads can increase your concurrency.</td>
</tr>
</tbody>
</table>
Vertica Consumer Settings

The following settings changes how Vertica acts when it consumes data from Kafka. You can set this value using the kafka_conf parameter on the KafkaSource UDL when directly executing a COPY statement. For schedulers, use the --message_max_bytes settings in the scheduler tool.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>message.max.bytes</td>
<td>Specifies the maximum size of a Kafka message. This size is the size of the JSON serialized message. To prevent truncated messages, set this value to the size of the largest possible message. This values should be the same on your sources, brokers, and producers.</td>
</tr>
</tbody>
</table>
Manually Copying Data From Kafka

You can manually stream data from Kafka into Vertica using a COPY statement. This technique is similar to copying data from other sources, such as the local file system, a client system, or from Apache Hadoop. See Bulk-Loading Data and the COPY statement reference for more information about using the COPY statement.

Manually copying data from Kafka is useful when you have a specific set of messages that you want to analyze. It is also useful when you want to explore the data in a Kafka stream before setting up a scheduler to continuously stream the data into Vertica.

Manually copying data from Kafka also gives you more control over the data load than using a scheduler. For example, suppose you want to perform business logic or custom rejection handling during the data load from Kafka. The scheduler does not support performing additional processing during its transactions. Instead, you may choose to periodically run a transaction that executes a COPY statement to load data from Kafka and then perform additional processing.

Unlike other copy methods, copying from Kafka usually implies your COPY statement loads data for a set period of time, rather than loading all of the data from a file or other source. For example, you can choose to COPY all of the messages sent to a Kafka topic for one minute. Vertica copies just the data streamed during that one minute period. After the duration is up, the COPY statement ends and Vertica does not load any further data.

**Important:** The durations you set for your data load is not exact. The duration actually controls how long the KafkaSource process runs. The actual COPY statement may run for a shorter period of time.

You can also choose specific ranges of messages to load from a topic using offsets. Kafka stores a backlog of messages for topics. How long it stores these messages is configured by the Kafka administrator. Copying using offsets lets you load this previously sent data. You can even choose to load all of the messages that Kafka has stored for a topic and also load the messages sent over a period of time.

When copying data from Kafka, the source of your COPY statement is always KafkaSource. Your COPY statement usually uses one of three parsers: KafkaParser, KafkaJSONParser, or KafkaAvroParser.

For example:

```sql
=> COPY public.from_kafka
       SOURCE KafkaSource(stream='iot_data|0|-2,iot_data|1|-2',
                           brokers='kafka01.example.com:9092',
                           duration=interval '10000 milliseconds')
```
In the previous example:

- The data is copied into a table named from_kafka in the public schema.
- The KafkaSource streams data from partitions 0 and 1 of the topic named iot_data.
- The streaming starts from the earliest available message in the stream set by the -2 in the stream offset parameter. This is a special offset value indicating that the load should start from the earliest possible message.
- KafkaSource reads the data from the Kafka broker running on the host named kafka01.example.com on port 9092.
- The streaming continues until either 10000 milliseconds (10 seconds).
- The stream is parsed as Avro data.
- The schema that the Avro parser uses to parse the data is retrieved from a schema registry running on the local system.
- Rejected data is saved in a table named public.rejections.

For more information, see the KafkaSource reference topic.

When you use COPY TRICKLE, Vertica recommends enabling the ReflexiveMoveout configuration parameter to trigger a Tuple Mover moveout task every time a commit occurs:

```sql
=> ALTER DATABASE dbname SET ReflexiveMoveout=1;
```

**Note:** If you are copying data containing default values into a flex table, you must identify the default value column as __raw__. For more information, refer to "Handling Default Values During Loading" in Bulk Loading Data into Flex Tables.

**Manually Loading Kafka Data Example**

This example demonstrates loading JSON-encoded data from a Kafka topic named web_hits that streams server logs of requests from a web site. The messages in the topic look like this:
In order to load this data, you first need a table to receive it. At first glance, the data seems rather uniform, so you could load it into a standard Vertica table. However, as this is JSON data, some messages may have added values that aren't apparent from a small sample. It is safer in this case to load the data into flex table. Flex tables can dynamically accept additional fields that appear in the data. For more information about flex tables, see Using Flex Tables.

The following example creates a flex table named web_hits to hold the Kafka data:

```sql
=> CREATE FLEX TABLE web_hits();
```

Before you load data from Kafka, you must determine the values for your call to the KafkaSource UDL based on your Kafka cluster and the topic you are loading:

- What are the host names (or IP addresses) and port numbers of the brokers in your Kafka cluster? The Kafka brokers are the service Vertica accesses in order to retrieve the Kafka data. In this example, the Kafka cluster has a single broker named kafka01.example.com, running on port 6667. This value is set in the brokers parameter.

- What topic will you be loading from, and from what partitions in that topic? Kafka topics are split their messages into different partitions to get scalable throughput. As mentioned earlier, this example will load data from a topic named web_hits. There is a single partition in web_hits, so this example will just specify 0 for the partition. This value is set in the stream parameter.

- What offset within the topic do you want to start loading data? Kafka keeps a backlog of messages. The Kafka administrator sets the length of this backlog. You can choose to load some or all of the messages in the backlog, or just load the currently streamed messages. You can also choose to end loading at a specific offset. This example will start loading data at the earliest possible message. The offset value to load the entire backlog is -2. This value is also set in the stream parameter.

- When should the COPY statement stop loading data? Copying data from Kafka is unlike most other sources (such as files), because streamed data is constantly arriving. You can choose to:
  - copy as much data as possible during a set amount of time.
  - have the COPY statement load data until no new data arrives within a timeout period.
load all of the data that is available, and not wait for any further data to arrive.

load a specific set of data from a starting offset to an ending offset

In this example, the COPY will run for 10000 milliseconds (10 seconds) to get a sample of the data. If the COPY statement is able to load the entire backlog of data in under 10 seconds, it will spend the remaining time loading streaming data as it arrives. This values is set in the duration parameter.

- How many threads will Vertica use to load the data from Kafka? Usually, you want to use 1 thread per partition being loaded from the Kafka topic. You can choose to not specify a value and let Vertica determine the number of threads to use based on the number of partitions and the resources available in the resource pool. This value is set in the KafkaSource's executionparallelism parameter. In this example, there is only one partition, so there's no need for additional threads to load data.

Note: The EXECUTIONPARALLELISM setting on the resource pool assigned to your COPY statement sets the upper limit on the number of threads it can use. Setting the executionparallelism parameter on the KafkaSource function call to a value that is higher than that of the resource pool's EXECUTIONPARALLELISM setting will not increase the number of threads Vertica uses beyond the limits of the resource pool.

- What parser do you need to parse the data? Kafka does not enforce any sort of formatting on the messages it sends. They are often in Avro or JSON format. However, they could be in any format. In this example, the data in the web_hits is encoded in JSON format, so it will use the KafkaJSONParser. This value is set in the COPY statement's PARSER clause.

- Where should rejected data be sent? Vertica saves raw Kafka messages that the parser cannot parse to a rejects table along with information on why it was rejected. This table is created by the COPY statement. This example will use the table named web_hits_rejections. This value is set in the COPY statement's REJECTED DATA AS TABLE clause.

The following statement loads the data using these options:

```sql
=> COPY web_hits SOURCE KafkaSource(stream='web_hits|0|-2',
   brokers='kafka01.example.com:6667',
   duration=interval '10000 milliseconds')
   PARSER KafkaJSONParser()
   REJECTED DATA AS TABLE public.web_hits_rejections DIRECT;
Rows Loaded
----------
20000
(1 row)
```

```sql
=> SELECT compute_flextable_keys('web_hits');
```
The stream parameter combines the name of the topic, the topic partition to load from, and the offset into a list delimited by a pipe character (|). To load from additional partitions in the same topic, or even additional topics, supply a comma separated list of topic name, partition number, and offset values delimited by pipe characters. For example, to load from partitions 0 through 2, your stream argument would be:  
stream='web_hits|0|-2,web_hits|1|-2,web_hits|2|-2'

**Note:** While you can load messages from different Kafka topics in the same COPY statement, you must ensure the data from the different topics is compatible with the target table's schema. The schema is less of a concern if you are loading data into a flex table, which can accommodate almost any data you want to load.

- The KafkaJSONParser accepts parameters that let you control how JSON data is transformed. See KafkaJSONParser for more information.
Automatically Copying Data From Kafka

Vertica offers a scheduler that loads streamed messages from one or more Kafka topics. Automatically loading streaming data has a number of advantages over manually using COPY:

- The streamed data automatically appears in your database. The frequency with which new data appears in your database is governed by the scheduler’s frame duration.

- The scheduler provides an exactly-once consumption process. The schedulers manage offsets for you so that each message sent by Kafka is consumed once.

- You can configure backup schedulers to provide high-availability. Should the primary scheduler fail for some reason, the backup scheduler automatically takes over loading data.

- The scheduler manages resources for the data load. You control its resource usage through the setting on the resource pool you assign to it. When loading manually, you must take into account the resources your load consumes.

There are a few drawbacks to using a scheduler which may make it unsuitable for your needs. You may find that schedulers do not offer the flexibility you need for your load process. For example, schedulers cannot perform business logic during the load transaction. If you need to perform this sort of processing, you are better off creating your own load process. This process would periodically run COPY statements to load data from Kafka. Then it would perform the business logic processing you need before committing the transaction.

For information on job scheduler requirements, refer to Vertica Integration for Apache Kafka.

What the Job Scheduler Does

The scheduler is responsible for scheduling loads of data from Kafka. The scheduler’s basic unit of processing is a frame, which is a period of time. Within each frame, the scheduler assigns a slice of time for each active microbatch to run. Each microbatch is responsible for loading data from a single source. Once the frame ends, the scheduler starts the next frame. The scheduler continues this process until you stop it.
The Anatomy of a Scheduler

Each scheduler has several groups of settings, each of which control an aspect of the data load. These groups are:

- The scheduler itself, which defines the configuration schema, frame duration, and resource pool.
- Clusters, which define the hosts in the Kafka cluster that the scheduler contacts to load data. Each scheduler can contain multiple clusters, allowing you to load data from multiple Kafka clusters with a single scheduler.
- Sources, which define the Kafka topics and partitions in those topics to read data from.
- Targets, which define the tables in Vertica that will receive the data. These tables can be traditional Vertica database tables, or they can be flex tables.
- Load specs, which define setting Vertica uses while loading the data. These settings include the parsers and filters Vertica needs to use to load the data. For example, if you are reading a Kafka topic that is in Avro format, your load spec needs to specify the Avro parser.
- Microbatches, which represent an individual segment of a data load from a Kafka stream. They combine the definitions for your cluster, source, target, and load spec that you create using the other vkconfig tools. The scheduler uses all of the information in the microbatch to execute COPY statements using the KafkaSource UDL function to transfer data from Kafka to Vertica. The statistics on each microbatch's load is stored in the stream_microbatch_history table.

The vkconfig Script

You use a Linux command-line script named vkconfig to create, configure, and run schedulers. This script is installed on your Vertica hosts along with the Vertica server in the following path:

/opt/vertica/packages/kafka/bin/vkconfig

Note: You can install and use the vkconfig utility on a non-Vertica host. You may want to do this if:
• You do not want the scheduler to use Vertica host resources.

• You want users who do not have shell accounts on the Vertica hosts to be able to set up and alter schedulers.

The easiest way to install vkconfig on a host is to install the Vertica server RPM. You must use the RPM that matches the version of Vertica installed on your database cluster. Do not create a database after installing the RPM. The vkconfig utility and its associated files will be in the `/opt/vertica/packages/kafka/bin` directory on the host.

The `vkconfig` script contains multiple tools. The first argument to the `vkconfig` script is always the tool you want to use. Each tool performs one function, such as changing one group of settings (such as clusters or sources) or starting and stopping the scheduler. For example, to create or configure a scheduler, you use the command:

```bash
$ /opt/vertica/packages/kafka/bin/vkconfig scheduler other options...
```

### What Happens When You Create a Scheduler

When you create a new scheduler, the `vkconfig` script takes the following steps:

• Creates a new Vertica schema using the name you specified for the scheduler. You use this name to identify the scheduler during configuration.

• Creates the tables needed to manage the Kafka data load in the newly-created schema. See [Data Streaming Schema Tables](#) for more information.

• Sets the LOCKTIMEOUT session configuration parameter to 0 for the session running the microbatches. When LOCKTIMEOUT is 0, data loads continuously because the scheduler does not have to wait for a lock to be released. If a table is already locked, Vertica cancels the frame and records an error in the events table.

• Creates the kafka_default_pool resource pool if it does not already exist. Schedulers use this resource pool if you do not assign them a pool of their own. Vertica recommends that you always create a resource pool for the exclusive use of your scheduler. See [Create a Resource Pool for Your Scheduler](#) for more information.
Validating Schedulers

When you create or configure a scheduler, it validates the following settings:

- Confirms that all brokers in the specified cluster exist.

- Connects to the specified host or hosts and retrieves the list of all brokers in the Kafka cluster. Getting this list always ensures that the scheduler has an up-to-date list of all the brokers. If the host is part of a cluster that has already been defined, the scheduler cancels the configuration.

- Confirms that the specified source exists.

- Retrieves the number of partitions in the source. If no number of partitions is specified, Vertica sets the value to the number of partitions the source has in the cluster.

You can disable validation using the --validation-type option in the vkconfig script's scheduler tool. See Scheduler Tool Options for more information.

Synchronizing Schedulers

By default, the scheduler automatically synchronizes its configuration and source information with Kafka host clusters. You can configure the synchronization interval using the --config-refresh scheduler utility option. Each interval, the scheduler:

- Checks for updates to the scheduler's configuration by querying its settings in its Vertica configuration schema.

- Performs all of the checks listed in Validating Schedulers.

You can configure synchronization settings using the --auto-sync option using the vkconfig script's scheduler tool. Scheduler Tool Options for details.

Launching a Scheduler

You use the vkconfig script's launch tool to launch a scheduler.
When you launch a scheduler, it collects data from your sources, starting at the specified offset. You can view the stream_microbatch_history table to see what the scheduler is doing at any given time.

To learn how to create, configure, and launch a scheduler, see Setting up a Scheduler in this guide.

You can also choose to bypass the scheduler. For example, you might want to do a single load with a specific range of offsets. For more information, see Manually Copying Data From Kafka in this guide.

## Managing a Running Scheduler

When you launch a scheduler from the command line, it runs in the foreground. It will run until you kill it (or the host shuts down). Usually, you want to start the scheduler as a daemon process that starts it when the host operating system starts, or after the Vertica database has started.

You shut down a running scheduler using the vkconfig script's shutdown tool. See Shutdown Tool Options for details.

You can change most of a scheduler’s settings (adding or altering clusters, sources, targets, and microbatches for example) while it is running. The scheduler automatically acts on the configuration updates.

##Launching Multiple Job Schedulers for High Availability

For high availability, you can launch two or more identical schedulers that target the same configuration schema. You differentiate the schedulers using the launch tool's `--instance-name` option (see Launch Tool Options). The scheduler not in use remains in stand-by mode until the active scheduler fails or is disabled. If the active scheduler fails, the backup scheduler takes over loading data from Kafka where the primary left off.
Viewing Schedulers from the MC

You can view the status of Kafka jobs from the MC. For more information, refer to Viewing Load History.

Updating Schedulers After Vertica Upgrades

A scheduler is only compatible with the version of Vertica that created it. Between Vertica versions, the scheduler's configuration schema or the UDx function the scheduler calls may change. You need to update your schedulers to account for these changes.

When you upgrade Vertica to a new major version or service pack, use the vkconfig scheduler tool's --upgrade option to update your scheduler. If you do not update a scheduler, you receive an error message if you try to launch it. For example:

```
$ vkconfig launch --conf weblog.conf
com.vertica.solutions.kafka.exception.FatalException: Configured scheduler schema and current scheduler configuration schema version do not match. Upgrade configuration by running:
  vkconfig scheduler --upgrade
  at com.vertica.solutions.kafka.scheduler.StreamCoordinator.assertVersion(StreamCoordinator.java:54)
  at com.vertica.solutions.kafka.scheduler.StreamCoordinator.run(StreamCoordinator.java:125)
  at com.vertica.solutions.kafka.Launcher.run(Launcher.java:285)
  at com.vertica.solutions.kafka.Launcher.main(Launcher.java:258)
$ vkconfig scheduler --upgrade --conf weblog.conf
Checking if UPGRADE necessary...
UPGRADE required, running UPGRADE...
UPGRADE completed successfully, now the scheduler configuration schema version is v8.1.1
$ vkconfig launch --conf weblog.conf
```

Setting up a Scheduler

You set up a scheduler using the Linux command line. Usually you perform the configuration on the host where you want your scheduler to run. It can be one of your Vertica hosts, or a separate host where you have installed the vkconfig utility (see The vkconfig Script for more information).

Follow these steps to set up and start a scheduler to stream data from Kafka to Vertica:
1. **Create a Config File (Optional)**
2. **Add the vkconfig Directory to Your Path (Optional)**
3. **Create a Resource Pool for Your Scheduler**
4. **Create the Scheduler**
5. **Create a Cluster**
6. **Create a Data Table**
7. **Create a Source**
8. **Create a Target**
9. **Create a Load-Spec**
10. **Create a Microbatch**
11. **Launch the Scheduler**

These steps are explained in the following sections. These sections will use the example of loading web log data (hits on a web site) from Kafka into a Vertica table.

### Create a Config File (Optional)

Many of the arguments you supply to the vkconfig script while creating a scheduler do not change. For example, you often need to pass a username and password to Vertica to authorize the changes to be made in the database. Adding the username and password to each call to vkconfig is tedious and error-prone.

Instead, you can pass the vkconfig utility a configuration file using the **--conf** option that specifies these arguments for you. It can save you a lot of typing and frustration.

The config file is a text file with a **keyword=value** pair on each line. Each keyword is a vkconfig command-line option, such as the ones listed in [Common vkconfig Script Options](#).

The following example shows a config file named `weblog.conf` that will be used to define a scheduler named `weblog_sched`. This config file is used throughout the rest of this example.

```bash
# The configuration options for the weblog_sched scheduler.
username=dbadmin
password=mypassword
dbhost=vertica01.example.com
```
Add the vkconfig Directory to Your Path (Optional)

The vkconfig script is located in the /opt/vertica/packages/kafka/bin directory. Typing this path for each call to vkconfig is tedious. You can add vkconfig to your search path for your current Linux session using the following command:

```bash
$ export PATH=/opt/vertica/packages/kafka/bin:$PATH
```

For the rest of your session, you are able to call vkconfig without specifying its entire path:

```bash
$ vkconfig
```

Valid options are scheduler, cluster, source, target, load-spec, microbatch, sync, launch, shutdown, help

If you want to make this setting permanent, add the export statement to your ~/.profile file. The rest of this example assumes that you have added this directory to your shell's search path.

Create a Resource Pool for Your Scheduler

Vertica recommends you always create a resource pool specifically for your scheduler. Schedulers assume they have exclusive use of the resource pool they are assigned. Using a separate pool for a scheduler lets you fine-tune its impact on your Vertica cluster's performance. You create resource pools within Vertica using the CREATE RESOURCE POOL statement.

```sql
=> CREATE RESOURCE POOL weblog_pool MEMORYSIZE '10%' PLANNEDCONCURRENCY 1 QUEUETIMEOUT 0;
```

One key setting in the resource pool for a scheduler is PLANNEDCONCURRENCY. This value tells Vertica how many simultaneous tasks (such as loads) will be using the resource pool. This value allows the scheduler to load from each topic simultaneously. This setting controls the number of simultaneous COPY statements the scheduler runs during each microbatch. Set this value to at least the number of topics the scheduler is reading data from. Setting it slightly higher than the number of topics is a good practice. The higher PLANNEDCONCURRENCY allocates resources for the scheduler's own internal processes, preventing them from impacting the data load.
Another important setting is the pool's EXECUTIONPARALLELISM which controls the number of threads that Vertica uses to load data from the topic. Ideally, you set this value to reflect the number of partitions within the topic. Often, a topic has too many partitions to assign a thread to each one. In these cases, you should set the EXECUTIONPARALLELISM to a value so the threads have an equal number of partitions to read from. For example, suppose the topic your scheduler is reading from has 100 partitions. Then you could set EXECUTIONPARALLELISM to 10, so that each thread will be reading from 10 partitions.

Finally, set the QUEUETIMEOUT parameter to 0. A value of 0 allows data to load continuously. If the scheduler has to wait for resources, it cannot progress, compromising scheduling configurations. By exclusively allocating a resource pool for your scheduler, you do not have to worry about it oversubscribing and causing queuing.

Leave reflexive moveout enabled. This option is on automatically when you create a scheduler. With reflexive moveout turned on, the Tuple Mover automatically performs a moveout operation when data is committed so that your WOS always has space to load data. For large volumes of data (>100 MB) use a load method of DIRECT.

Not allocating enough resources to your schedulers can result in errors. For example, you may get OVERSHOT DEADLINE FOR FRAME errors if the scheduler is not able to load data from all of the topics it is supposed to in a data frame.

If you do not create and assign a resource pool for your scheduler, it will use the default resource pool named kafka_default_pool.

See Resource Pool Architecture for more information about resource pools.

Create the Scheduler

Vertica includes a default scheduler named stream_config. You can use this scheduler or create a new scheduler using the vkconfig script's scheduler tool with the --create and --config-schema options:

$ vkconfig scheduler --create --config-schema scheduler_name --conf conf_file

The --create and --config-schema options are the only ones required to add a scheduler with default options. This command creates a new schema in Vertica that holds the scheduler's configuration. See What Happens When You Create a Scheduler for details on the creation of the scheduler's schema.

You can use additional configuration parameters to further customize your scheduler. See Scheduler Tool Options for more information.

The following example:
- Creates a scheduler named weblog_sched using the --config-schema option.

- Grants privileges to configure and run the scheduler to the Vertica user named kafka_user with the --operator option. The dbadmin user must specify additional privileges separately.

- Specifies a frame duration of thirty seconds with the --frame-duration option.

- Sets the resource pool that the scheduler uses to the weblog_pool created earlier:

  ```
  $ vkconfig scheduler --create --config-schema weblog_sched --operator kafka_user \
  --frame-duration '00:00:30' --resource-pool weblog_pool --conf weblog.conf
  ```

  **Note:** Technically, the previous example doesn't need to supply the --config-schema argument because it is set in the weblog.conf file. It appears in this example for clarity. There's no harm in supplying it on the command line as well as in the configuration file, as long as the values match. If they do not match, the value given on the command line takes priority.

---

### Create a Cluster

You must associate at least one Kafka cluster with your scheduler. Schedulers can access more than one Kafka cluster. To create a cluster, you supply a name for the cluster and host names and ports the Kafka cluster's brokers.

When validating your settings, the scheduler connects to the cluster and automatically retrieves the list of all brokers in the cluster. Therefore, you do not have to list every single broker in the --hosts parameter.

The following example creates a cluster named kafka_weblog, with two Kafka broker hosts: kafka01 and kafka03 in the example.com domain. The Kafka brokers are running on port 9092.

  ```
  $ vkconfig cluster --create --cluster kafka_weblog \
  --hosts kafka01.example.com:9092,kafka03.example.com:9092 --conf weblog.conf
  ```

See [Cluster Tool Options](#) for more information.
Create a Source

Next, create at least one source for your scheduler to read. The source defines the Kafka topic the scheduler loads data from as well as the number of partitions the topic contains.

To create and associate a source with a configured scheduler, use the source tool. When you create a source, Vertica connects to the Kafka cluster to verify that the topic exists. So, before you create the source, make sure that the topic already exists in your Kafka cluster. Because Vertica verifies the existence of the topic, you must supply the previously-defined cluster name using the --cluster option.

The following example creates a source for the Kafka topic named web_hits on the cluster created in the previous step. This topic has a single partition.

```
$ vkconfig source --create --cluster kafka_weblog --source web_hits --partitions 1 --conf weblog.conf
```

Note: The --partitions parameter is the number of partitions to load, not a list of individual partitions. For example, if you set this parameter to 3, the scheduler will load data from partitions 0, 1, 2, and 3.

See Source Tool Options for more information.

Create a Data Table

Before you can create a target for your scheduler, you must create a target table in your Vertica database. This is the table Vertica uses to store the data the scheduler loads from Kafka. You must decide which type of table to create for your target:

- A standard Vertica database table, which you create using the CREATE TABLE statement. This type of table stores data efficiently. However, you must ensure that its columns match the data format of the messages in Kafka topic you are loading. You cannot load complex types of data into a standard Vertica table.

- A flex table, which you create using CREATE FLEX TABLE. A flex table is less efficient than a standard Vertica database table. However, it is flexible enough to deal with data whose schema varies and changes. It also can load most complex data types that (such as maps and lists) that standard Vertica tables cannot.

**Important:** Avoid having columns with primary key restrictions in your target table. The scheduler stops loading data if it encounters a row that has a value which violates this
restriction. If you must have a primary key restricted column, try to filter out any redundant values for that column in the streamed data before it is loaded by the scheduler.

The data in this example is in a set format, so the best table to use is a standard Vertica table. The following example creates a table named web_hits to hold four columns of data. This table is located in the public schema.

```
=> CREATE TABLE web_hits (ip VARCHAR(16), url VARCHAR(256), date DATETIME, user_agent VARCHAR(1024));
```

Note: You do not need to create a rejection table to store rejected messages. The scheduler creates the rejection table automatically.

Create a Target

Once you have created your target table, you can create your scheduler’s target. The target tells your scheduler where to store the data it retrieves from Kafka. This table must exist when you create your target. You use the vkconfig script's target tool with the --target-schema and --target-table options to specify the Vertica target table's schema and name. The following example adds a target for the table created in the previous step.

```
$ vkconfig target --create --target-schema public --target-table web_hits --conf weblog.conf
```

See Target Tool Options for more information.

Create a Load Spec

The scheduler's load spec provides parameters that Vertica uses when parsing the data loaded from Kafka. The most important option is --parser which sets the parser that Vertica uses to parse the data. You have three parser options:

- **KafkaAvroParser** for data in Avro format.
- **KafkaJSONParser** for data in JSON format.
- **KafkaParser** to load each message into a single VARCHAR field. See Parsing Custom Formats for more information.

In this example, the data being loaded from Kafka is in JSON format. The following command creates a load spec named weblog_load and sets the parser to KafkaJSONParser.

```
$ vkconfig load-spec --create --parser KafkaJSONParser --load-spec weblog_load --conf weblog.conf
```
See Load Spec Tool Options for more information.

Create a Microbatch

The microbatch combines all of the settings added to the scheduler so far to define the individual COPY statements that the scheduler uses to load data from Kafka.

The following example uses all of the settings created in the previous examples to create a microbatch called weblog.

```
$ vkconfig microbatch --create --microbatch weblog --target-schema public --target-table web_hits --add-source web_hits --add-source-cluster kafka_weblog --load-spec weblog_load --conf weblog.conf
```

See Microbatch Tool Options for more information.

Launch the Scheduler

Once you've created at least one microbatch, you can run your scheduler. You start your scheduler using the launch tool, passing it the name of the scheduler's schema. The scheduler begins scheduling microbatch loads for every enabled microbatch defined in its schema.

The following example launches the weblog scheduler defined in the previous steps.

```
$ vkconfig launch --conf weblog.conf &
```

Important: Vertica does not recommend specifying a password on the command line. Passwords on the command line can be exposed by the system's list of processes, which shows the command line for each process. Instead, put the password in a configuration file. Make sure the configuration file's permissions only allow it to be read by the user.

See Launch Tool Options for more information.

Checking that the Scheduler is Running

Once you have launched your scheduler, you can verify that it is running by querying the stream_microbatch_history table in the scheduler's schema. This table lists the results of each microbatch the scheduler has run.

For example, this query lists the microbatch name, the start and end times of the microbatch, the start and end offset of the batch, and why the batch ended. The results are ordered to start from when the scheduler was launched:
#### Limiting Loads Using Offsets

Kafka maintains a user-configurable backlog of messages. By default, a newly-created scheduler reads all of the messages in a Kafka topic, including all of the messages in the backlog, not just the messages that are streamed out after the scheduler starts. Often, this is what you want.

In some cases, however, you may want to stream just a section of a source into a table. For example, suppose you want to analyze the web traffic of your e-commerce site starting at specific date and time. However, your Kafka topic contains web access records from much further back in time than you want to analyze. In this case, you can use an offset to stream just the data you want into Vertica for analysis.

Another common use case is when you have already loaded data some from Kafka manually (see [Manually Copying Data From Kafka](https://docs.vertica.com/en-US/9.0.0/Vertica-Analytic-Database-User-Guide/manually-copying-data-from-kafka)). Now you want to stream all of the newly-arriving data. By default, your scheduler will reload all of the previously loaded data (assuming it is still available from Kafka). You can use an offset to tell your scheduler to start automatically loading data at the point where your manual data load left off.
Configuring a Scheduler to Start Streaming From an Offset

The `vkconfig` script's microbatch tool has an `--offset` option that lets you specify the index of the message in the source where you want the scheduler to begin loading. This option accepts a comma-separated list of index values. You must supply one index value for each partition in the source unless you use the `--partition` option. This option lets you choose the partitions the offsets apply to. The scheduler cannot be running when you set an offset in the microbatch.

If your microbatch defines more than one cluster, use the `--cluster` option to select which one the offset option applies to. Similarly, if your microbatch has more than one source, you must select one using the `--source` option.

For example, suppose you want to load just the last 1000 messages from a source named `web_hits`. To make things easy, suppose the source contains just a single partition, and the microbatch defines just a single cluster and single topic.

Your first task is to determine the current offset of the end of the stream. You can do this on one of the Kafka nodes by calling the `GetOffsetShell` class with the time parameter set to `-1` (the end of the topic):

```bash
$ path to kafka/bin/kafka-run-class.sh kafka.tools.GetOffsetShell \
    --broker-list kafka01:9092,kafka03:9092 --time -1 \
    --topic web_hits
{metadata.broker.list=kafka01:9092,kafka03:9092, request.timeout.ms=1000, 
client.id=GetOffsetShell, security.protocol=PLAINTEXT}
web_hits:0:8932
```

You can also use `GetOffsetShell` to find the offset in the stream that occurs before a timestamp.

In the above example, the `web_hits` topic's single partition has an ending offset of 8932. If we want to load the last 1000 messages from the source, we need to set the microbatch's offset to 8932 - 1001 or 7931.

**Note:** The start of an offset is inclusive in the Vertica COPY statement. Kafka's native starting offset is exclusive. Therefore, you must add one to the offset to get the correct number of messages.

With the offset calculated, you are ready to set it in the microbatch's configuration. The following example:
- Shuts down the scheduler whose configuration information stored in the weblog.conf file.
- Sets the starting offset using the microbatch utility.
- Restarts the scheduler.

```
$ vkconfig shutdown --conf weblog.conf
$ vkconfig microbatch --microbatch weblog --update --conf weblog.conf --offset 7931
$ vkconfig launch --conf weblog.conf &
```

If the target table was empty or truncated before the scheduler started, it will have 1000 rows in the table in it (until more messages are streamed through the source):

```
=> select count(*) from web_hits;
   count
  -------
   1000
(1 row)
```

**Note:** The last example assumes that the offset values for the last 1000 messages in the Kafka topic were assigned consecutively. This assumption is not always be true. A Kafka topic can have gaps in its offset numbering for a variety of reasons. Offsets refer to the key value assigned to a message by Kafka, not its position in the topic.

### Ending a COPY From Kafka at an Offset

When you manually copy data from a Kafka source into Vertica, the COPY statement defaults to loading all of the messages in the topic. You can specify a maximum offset the COPY should load to by supplying an end offset value in addition to the starting offset in the KafkaSource's source parameter.
Parsing Custom Formats

Vertica supports the use of user-defined filters to manipulate data arriving from your streaming message bus. You can apply these filters to data before you parse it. By default, data that flows from the source does not contain message boundaries. The default Kafka parsers can locate the message boundaries on their own. However, other user-defined and Vertica parsers are not designed to recognize the message boundaries. Filters let you insert message boundary information to the data stream (such as delimiters) so that the parser are able to extract and parse individual messages.

Filters for Use with Kafka Data

Vertica includes the following filters:

- KafkaInsertDelimiters — Transforms the Kafka data stream by inserting a user-specified delimiter between each record. The delimiter can contain any characters and be of any length. This parser uses the following syntax:

  \[\text{KafkaInsertDelimiters}(\text{delimiter} = \text{'delimiter'})\]

- KafkaInsertLengths — Transforms the Kafka data stream by inserting the length of the following record in bytes at the beginning of the record. Vertica writes lengths as 4-byte uint32 values in Big Endian network byte order. For example, a 100-byte record would be preceded by 0x00000064.

  \[\text{KafkaInsertLengths()}\]

**Note:** The Vertica provided filters are mutually exclusive. You cannot use both to process a Kafka data stream in the same COPY statement.

Vertica also supports the use of additional Vertica and user-defined filters. If you are using a Vertica filter, it must appear first in the filter list. Use a comma to delimit multiple filters. If you are using a non-Kafka parser, you must use at least one filter to prepare your content for that parser. If you do not provide a filter, the parser fails with the message:

**Input is not from Kafka source.**
Examples

The following example shows how you can delimit data streams from two hosts by the string \\n. You can then use a CSV parser to parse the content.

```sql
=> COPY stream_data.target_table SOURCE KafkaSource (stream='source1|1|1,source2|2|2',
brokers='host1:9092,host2:9092',
duration= INTERVAL '10s')
FILTER KafkaInsertDelimiters(delimiter = '\n')
PARSER MyCsvParser(recordTerminator = '\n');
```

The following example shows how you can specify that a Vertica filter and a decryption filter process a single Kafka data stream. Using the length information the KafkaInsertLengths filter injects, the parser can identify each record and parse it individually.

```sql
=> COPY stream_data.target_table SOURCE KafkaSource (stream='source1|1|1, brokers='host1:9092')
FILTER KafkaInsertLengths() DecryptFilter(parameter=Key)
PARSER ComplexParser(parameter = 'value');
```
Using a Schema Registry with Kafka

Vertica supports the use of a Confluent schema registry for Avro schemas with the `KafkaAvroParser`. By using a schema registry, you enable the Avro parser to parse and decode messages written by the registry and to retrieve schemas stored in the registry. In addition, a schema registry enables Vertica to process streamed data without sending a copy of the schema with each record. Vertica can access a schema registry in the following ways:

- schema ID
- subject and version

Note: If you use the compatibility config resource in your schema registry, you should specify a value of at least BACKWARD. You may also choose to use a stricter compatibility setting. For more information on installing and configuring a schema registry, refer to the Confluent documentation.

Schema ID Loading

In schema ID based loading, the Avro parser checks the schema ID associated with each message to identify the correct schema to use. A single COPY statement can reference multiple schemas. Because each message is not validated, Vertica recommends that you use a flex table as the target table for schema ID based loading.

The following example shows a COPY statement that refers to a schema registry located on the same host.

```sql
=> COPY logs source kafkasource(stream='simple|0|0', stop_on_eof=true, duration=interval '10 seconds') parser KafkaAvroParser(schema_registry_url='http://localhost:8081/');
```

Subject and Version Loading

In subject and version loading, you specify a subject and version in addition to the schema registry URL. The addition of the subject and version identifies a single schema to use for all messages in the COPY. If any message in the statement is incompatible with the schema, the COPY fails. Because all messages are validated prior to loading, Vertica recommends that you use a standard Vertica table as the target for subject and version loading.
The following example shows a COPY statement that identifies a schema subject and schema version as well as a schema registry.

```sql
=> COPY t source kafkasource(stream='simpleEvolution|0|0',
stop_on_eof=true, duration=interval '10 seconds') parser
KafkaAvroParser(schema_registry_url='http://repository:8081/schema-repo/',
schema_registry_subject='simpleEvolution-value',schema_registry_version='1')
REJECTED DATA AS TABLE "t_rejects";
```
Using Vertica to Produce Data for Kafka

In addition to consuming data from Kafka, Vertica can produce data for Kafka. There are two ways you can have Vertica stream data to Kafka:

- Using the KafkaExport function to send Vertica data to a Kafka topic. Use this feature to export arbitrary data to Kafka for consumption by any other application that is a Kafka consumer.

- Using notifiers to send Vertica diagnostic data from the data collector tables. You use this method primarily to let third-party monitoring applications view the status of your Vertica cluster. See Monitoring Vertica Using Notifiers.

The topics in this section explain how to use these methods in greater detail.

Producing Data Using KafkaExport

The KafkaExport function lets you stream data from Vertica to Kafka. You pass this function three arguments and two or three parameters:

```
SELECT KafkaExport(partitionColumn, keyColumn, valueColumn
        USING PARAMETERS brokers='host[:port][,host...]',
        topic='topicname'
          [,kafka_conf='kafka_configuration_setting']
    OVER (partition_clause) FROM table;
```

The `partitionColumn` and `keyColumn` arguments set the Kafka topic's partition and key value, respectively. You can set either or both of these values to NULL. If you set the partition to NULL, Kafka uses its default partitioning scheme (either randomly assigning partitions if the key value is NULL, or based on the key value if it is not).

The `valueColumn` argument is a LONG VARCHAR containing message data that you want to send to Kafka. Kafka does not impose structure on the message content. Your only restriction on the message format is what the consumers of the data are able to parse.

You are free to convert your data into a string in any way you like. For simple messages (such as a comma-separated list), you can use functions such as CONCAT to assemble your values into a message. If you need a more complex data format, such as JSON, consider writing a UDx function that accepts columns of data and outputs a LONG VARCHAR containing the data in the format you require. See Developing User-Defined Extensions (UDxs) for more information.

See KafkaExport for detailed information about KafkaExport's syntax.
Export Example

This example shows you how to perform a simple export of several columns of a table. Suppose you have the following table containing a simple set of Internet of things (IOT) data:

```
=> SELECT * FROM iot_report LIMIT 10;
server | date       | location | id
-------+------------+----------+-------
1      | 2016-10-11 | -14.86058, 112.75848 | 70982027
1      | 2017-07-02 | -21.42197, -127.17672 | 49494918
1      | 2017-10-19 | -71.72156, -36.27381 | 94328189
1      | 2018-07-11 | -9.41315, 102.36866 | 48366610
1      | 2018-08-30 | 83.97962, 162.83848 | 967212
2      | 2017-01-20 | 37.17372, 136.14026 | 36670100
2      | 2017-07-29 | -38.99517, 171.72671 | 52049116
2      | 2018-04-19 | 69.28989, 133.98275 | 36059026
2      | 2018-08-28 | -59.71784, -144.97142 | 77310139
2      | 2018-09-14 | 58.07275, 111.07354 | 4198109
(10 rows)
```

You want to send the data in this table to a Kafka topic named iot_results for consumption by other applications. Looking at the data and the structure of the iot_report, you may decide the following:

- The server column is a good match for the partitions in iot_report. There are three partitions in the Kafka topic, and the values in server column are between 1 and 3. Suppose the partition column had a larger range of values (for example, between 1 and 100). Then you could use the modulo operator (%) to coerce the values into the same range as the number of partitions (server % 3). A complication with these values is that the values in the server are 1-based (the lowest value in the column is 1). Kafka's partition numbering scheme is zero-based. So, you must adjust the values in the server column by subtracting 1 from them.

- The id column can act as the key. This column has a data type of INTEGER. The KafkaExport function expects the key value to be a VARCHAR. Vertica does not automatically cast INTEGER values to VARCHAR, so you must explicitly cast the value in your function call.
The consumers of the `iot_report` topic expect values in comma-separated format. You can combine the values from the date and location columns into a single `VARCHAR` using nested calls to the `CONCAT` function.

The final piece of information you need to know is the host names and port numbers of the brokers in your Kafka cluster. In this example, there are two brokers named `kafka01` and `kafka03`, running on port 6667 (the port that Hortonworks clusters use). Once you have all of this information, you are ready to export your data.

The following example shows how you might export the contents of `iot_report`:

```sql
=> SELECT KafkaExport(server - 1, id::VARCHAR,
    CONCAT(CONCAT(date, ', '), location)
 USING PARAMETERS brokers='kafka01:6667,kafka03:6667',
    topic='iot_results') OVER (PARTITION BEST) FROM iot_report;
```

<table>
<thead>
<tr>
<th>partition</th>
<th>key</th>
<th>message</th>
<th>failure_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0 rows)

KafkaExport returned 0 rows which means Vertica was able to send all of your data to Kafka without any errors.

Other things to note about the example:

- The `CONCAT` function automatically converts the date column's `DATETIME` value to a `VARCHAR` for you, so you do not need to explicitly cast it.

- Two nested `CONCAT` functions are necessary to concatenate the date field with a comma, and the resulting string with the location field.

- Adding a third column to the message field would require two additional `CONCAT` function calls (one to concatenate a comma after the location column, and one to concatenate the additional column's value). Using `CONCAT` becomes messy after just a few column's worth of data.

**KafkaExport's Return Value**

KafkaExport outputs any rows that Kafka rejected. For example, suppose you forgot to adjust the partition column to be zero-based in the previous example. Then some of the rows exported to Kafka would specify a partition that does not exist. In this case, Kafka rejects these rows, and KafkaExport reports them in table format:

```sql
=> SELECT KafkaExport(server, id::VARCHAR,
    CONCAT(CONCAT(date, ', '), location)
 USING PARAMETERS brokers='kafka01:6667,kafka03:6667',
```
You can capture this output by creating a table to hold the rejects. Then use an INSERT statement to insert KafkaExport's results:

```sql
=> CREATE TABLE export_rejects (partition INTEGER, key VARCHAR, message LONG VARCHAR, failure_reason VARCHAR);
=> CREATE TABLE
=> INSERT INTO export_rejects SELECT KafkaExport(server, id::VARCHAR,
    CONCAT(CONCAT(date, ',', location))
    USING PARAMETERS brokers='kafka01:6667,kafka03:6667',
    topic='iot_results') OVER (PARTITION BEST) FROM iot_report;
```

```
<table>
<thead>
<tr>
<th>partition</th>
<th>key</th>
<th>message</th>
<th>failure_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>48492866</td>
<td>2017-10-10 12:08:33, 78.84883, -13.79584</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>3</td>
<td>73846006</td>
<td>2017-10-06 16:50:57, -25.33024, -157.91389</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>3</td>
<td>27462612</td>
<td>2018-08-19 00:02:18, 13.00436, 85.44815</td>
<td>Local: Unknown partition</td>
</tr>
</tbody>
</table>
```

(4 rows)

You can capture this output by creating a table to hold the rejects. Then use an INSERT statement to insert KafkaExport's results:

```
=> CREATE TABLE export_rejects (partition INTEGER, key VARCHAR, message LONG VARCHAR, failure_reason VARCHAR);
=> CREATE TABLE
=> INSERT INTO export_rejects SELECT KafkaExport(server, id::VARCHAR,
    CONCAT(CONCAT(date, ',', location))
    USING PARAMETERS brokers='kafka01:6667,kafka03:6667',
    topic='iot_results') OVER (PARTITION BEST) FROM iot_report;
```

```
<table>
<thead>
<tr>
<th>partition</th>
<th>key</th>
<th>message</th>
<th>failure_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>27462612</td>
<td>2018-08-19 00:02:18, 13.00436, 85.44815</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>3</td>
<td>48492866</td>
<td>2017-10-10 12:08:33, 78.84883, -13.79584</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>3</td>
<td>73846006</td>
<td>2017-10-06 16:50:57, -25.33024, -157.91389</td>
<td>Local: Unknown partition</td>
</tr>
</tbody>
</table>
```

(4 rows)
Using SSL with Kafka

Vertica supports the use of SSL authentication between Kafka, Vertica, and the Kafka Scheduler. For information on SSL authentication with Vertica, refer to TLS/SSL Server Authentication.
Scheduler/Vertica Communication

The scheduler uses keystore information on a per-session basis to communicate with Vertica. It passes information through a JDBC SSL connection to Vertica. The Scheduler connects to Vertica through a JVM. As a result, your SSL keys must be within the JVM keystore and the CA must be within the JVM truststore. When connecting between Vertica and Kafka, the Scheduler uses its keys to authenticate Kafka.

Vertica/Kafka Communication

In a typical configuration, each Kafka broker contains its own key store and trust store. Vertica and Kafka authenticate against your certifying authority. Kafka authenticates by means of the librdkafka library.

Adding SSL to Vertica/Kafka Integration

To add SSL authentication to your Vertica/Kafka integration, you must configure SSL support on various components in Vertica, Kafka, and Java.

1. Configure Vertica for SSL.
2. Configure Kafka for SSL.
3. Configure your scheduler for SSL.
4. Configure Java on your scheduler for SSL.

Configuring Kafka for SSL

A typical Kafka/Vertica configuration consists of the following steps:

1. Create a certifying authority certificate.
2. Create a trust store and key store on your Kafka brokers using your certificate.
3. Configure the server.properties file on each Kafka broker to use your key store.

Create a Certifying Authority Certificate

If you do not already have one, create a certifying authority certificate, as shown in the following example.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

For more information on generating a certificate, refer to the OpenSSL documentation.

Create a Trust Store and Key Store on your Kafka Brokers

You must create a trust store and key store on each of your Kafka brokers.

1. Create a trust store on your Kafka broker by importing your certificate. The following example shows a typical trust store command.

```
keytool -keystore kafka.truststore.jks -alias caroot -import -file ca-cert
```

2. Create a key store on your Kafka broker, as shown in the following example. The following example shows a typical series of key store commands.

```
keytool -keystore brokernamem keystore.jks -alias brokernamem -validity 365 -keyalg RSA -genkey
keytool -keystore brokernamem keystore.jks -alias brokernamem -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreateserial
keytool -keystore brokernamem -alias caroot -import -file ca-cert
keytool -keystore brokernamem keystore.jks -alias brokernamem$1 -import -file cert-signed
```

3. Repeat the key store and trust store configuration process for each Kafka broker, specifying the correct alias and key store name for that broker.

Configure the server.properties File on Each Kafka Broker to Use Your Key Store

You can pass SSL configuration information from your Kafka broker to Vertica by including user defined session parameters in your server.properties file. For more information on configuring your server.properties file, refer the Apache Kafka documentation.
**Configuring Your Scheduler for SSL**

Your scheduler requires a trust store, key store and launch parameters to use SSL.

### Create a Trust Store and and Key Store on Your Scheduler

You must create a trust store and key store on your scheduler.

1. Create a trust store on your scheduler by importing your certificate. The following example shows a typical trust store command.

   ```bash
   keytool -keystore schedulername.truststore.jks -alias caroot -import -file ca-cert
   ```

2. Create a key store on your scheduler, as shown in the following example. The following example shows a typical series of key store commands.

   ```bash
   keytool -keystore schedulername.keystore.jks -alias vsched -validity 365 -keyalg RSA -genkey
   keytool -keystore schedulername.keystore.jks -alias vsched -certreq -file cert-file
   openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365
   keytool -keystore schedulername.keystore.jks -alias caroot -import -file ca-cert
   keytool -keystore schedulername.keystore.jks -alias vsched -import -file cert-signed
   ```

### Launch Your Scheduler with SSL Enabled

When you launch your scheduler, you must include SSL parameters to enable SSL support. For more information on scheduler utility parameters, refer to Scheduler Tool Options.

The following example shows a launch command including all the required SSL parameters

```bash
/opt/vertica/packages/kafka/bin/vkconfig launch --enable-SSL true --ssl-ca-alias authentificcert --ssl-key-alias ourkey --ssl-key-password secret
```
Configuring Java to Use SSL

The scheduler uses JDBC to communicate with your Vertica session. As a result, you must configure your scheduler JVM and JDBC settings for SSL. For information on configuring your JVM and JDBC, refer to the Java documentation.

Configure Your Scheduler JVM

To use SSL authentication with Kafka, you must configure your scheduler JVM to use the key store and trust store you created. Vertica recommends that you use a configuration file to provide this information. Include the following options in your configuration file:

-Djavax.net.ssl.trustStore=/path/to/truststore
-Djavax.net.ssl.keyStore=/path/to/keystore
-Djavax.net.ssl.keyStorePassword=password1234

Configure Your Scheduler JDBC to use SSL

Add the following to your scheduler's configuration:

--jdbc-opt ssl=true

This option add the ssl=true argument to the scheduler's JDBC connection string.

Using SSL Without a Scheduler

You can also configure Vertica to interact with Kafka without using a scheduler. In this case, Kafka integration requires user-defined extensions (UDXs) that do not have access to global Vertica configuration parameters. As a result, you must load your SSL Certificates as session parameters.

1. Configure your MaxSessionUDParameterSize to a value larger than the length of your certificate chain.

ALTER SESSION SET MaxSessionUDParameterSize=100000
2. Provide certificate, key, and password information to your session. Vertica does not log parameters that end in _secret. For more information on these parameters, refer to User-Defined Session Parameters.

```
=> ALTER SESSION SET UDPARAMETER kafka_SSL_Certificate='<client.crt contents>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKey_secret='<client.key contents>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_PrivateKeyPassword_secret='<password, if applicable>';  
ALTER SESSION SET UDPARAMETER kafka_SSL_CA='<ca.crt contents>';  
```

3. Enable SSL authentication for your session.

```
=> ALTER SESSION SET kafka_Enable_SSL=1;  
```

You can now run COPY commands with SSL authentication.
Using kafkacat to Troubleshoot Kafka Integration Issues

Kafkacat is a third-party open-source utility that lets you connect to Kafka from the Linux command line. It uses the same underlying library that the Vertica integration for Apache Kafka uses to connect to Kafka. This shared library makes kafkacat a useful tool for testing and debugging your Vertica integration with Kafka.

You may find kafkacat useful for:

- Testing connectivity between your Vertica and Kafka clusters.
- Examining Kafka data for anomalies that may prevent some of it from loading into Vertica.
- Producing data for test loads into a Vertica.
- Listing details about a Kafka topic.

For more information about kafkacat, see its project page at Github.

Running kafkacat on Vertica Nodes

The kafkacat utility is bundled in the Vertica install package, so it is available on all nodes of your Vertica cluster in the /opt/vertica/packages/kafka/bin directory. This is the same directory containing the vkconfig utility, so if you have added it to your path, you can use the kafkacat utility without specifying its full path. Otherwise, you can add this path to your shell's environment variable using the command:

```bash
set PATH=/opt/vertica/packages/kafka/bin:$PATH
```

Note: On Debian and Ubuntu systems, you must tell kafkacat to use Vertica's own copy of the SSL libraries by setting the LD_LIBRARY_PATH environment variable:

```bash
$ export LD_LIBRARY_PATH=/opt/vertica/lib
```

If you do not set this environment variable, the kafkacat utility exits with the error:

```
kafkacat: error while loading shared libraries: libcrypto.so.10: cannot open shared object file: No such file or directory
```

Executing kafkacat without any arguments gives you a basic help message:
$ kafkacat
Error: -b <broker,..> missing

Usage: kafkacat options> [file1 file2 .. | topic1 topic2 ..]]
kafkacat - Apache Kafka producer and consumer tool
https://github.com/edenhill/kafkacat
Copyright (c) 2014-2015, Magnus Edenhill
Version releases/VER_8_1_RELEASE_BUILD_1_555_20170615-4931-g3fb918 (librdkafka releases/VER_8_1_RELEASE_BUILD_1_555_20170615-4931-g3fb918)

General options:
- C | -P | -L Mode: Consume, Produce or metadata List
- G <group-id> Mode: High-level KafkaConsumer (Kafka 0.9 balanced consumer groups)
  Expects a list of topics to subscribe to
- t <topic> Topic to consume from, produce to, or list
- p <partition> Partition
- b <brokers,..> Bootstrap broker(s) (host[:port])
- D <delim> Message delimiter character:
  a-z.. | \r | \n | \t | \xNN
  Default: \n
- K <delim> Key delimiter (same format as -D)
- c <cnt> Limit message count
- X list List available librdkafka configuration properties
- X prop=val Set librdkafka configuration property.
  Properties prefixed with "topic." are applied as topic properties.
- X dump Dump configuration and exit.
- d <dbg1,...> Enable librdkafka debugging:
  all,generic,broker,topic,metadata,queue,msg,protocol,cgrp,security,fetch,feature
- q Be quiet (verbosity set to 0)
- v Increase verbosity
- V Print version

Producer options:
- z snappy|gzip Message compression. Default: none
- p -1 Use random partitioner
- D <delim> Delimiter to split input into messages
- K <delim> Delimiter to split input key and message
- l Send messages from a file separated by delimiter, as with stdin.
  (only one file allowed)
- T Output sent messages to stdout, acting like tee.
- c <cnt> Exit after producing this number of messages
- Z Send empty messages as NULL messages
  file1 file2.. Read messages from files.
  With -l, only one file permitted.
  Otherwise, the entire file contents will be sent as one single message.

Consumer options:
- o <offset> Offset to start consuming from:
  beginning | end | stored |
  <value> (absolute offset) |
  -<value> (relative offset from end)
- e Exit successfully when last message received
- f <fmt..> Output formatting string, see below.
  Takes precedence over -D and -K.
- D <delim> Delimiter to separate messages on output
- K <delim> Print message keys prefixing the message
Testing Connectivity to a Kafka Cluster and Getting Metadata

One basic troubleshooting step you often need to perform is verifying that Vertica nodes can connect to the Kafka cluster. Successfully executing just about any kafkacat command will prove the Vertica node you are logged into is able to reach the Kafka cluster. One simple command you can execute to verify connectivity is to get the metadata for all of the topics the Kafka cluster has defined. The following example demonstrates using kafkacat’s metadata

```bash
kafka -b <broker> -t <topic> -p <partition>
```

or:

```bash
kafka -C -b ...
```

High-level KafkaConsumer mode:

```bash
kafka -b <broker> -G <group-id> topic1 top2 ^aregex\d+
```

Producer mode (reads messages from stdin):

```bash
... | kafka -b <broker> -t <topic> -p <partition>
```

or:

```bash
kafka -P -b ...
```

Metadata listing:

```bash
kafka -L -b <broker> [-t <topic>]
```
listing command to connect to the broker named kafka01 running on port 6667 (the Kafka broker port used by Hortonworks Hadoop clusters).

```
$ kafkacat -L -b kafka01:6667

Metadata for all topics (from broker -1: kafka01:6667/bootstrap):
  2 brokers:
    broker 1001 at kafka03.example.com:6667
    broker 1002 at kafka01.example.com:6667
  4 topics:
    topic "iot-data" with 3 partitions:
      partition 2, leader 1002, replicas: 1002, isrs: 1002
      partition 1, leader 1001, replicas: 1001, isrs: 1001
      partition 0, leader 1002, replicas: 1002, isrs: 1002
    topic "__consumer_offsets" with 50 partitions:
      partition 23, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 41, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 32, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 8, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 17, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 44, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 35, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 26, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 11, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 29, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 38, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 47, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 20, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 2, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 5, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 14, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 46, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 49, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 40, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 4, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 13, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 22, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 31, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 16, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 7, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 43, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 25, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 34, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 10, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 37, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 1, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 19, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 28, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 45, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 36, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 27, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 9, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 18, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 21, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 48, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 12, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 3, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 30, leader 1002, replicas: 1002,1001, isrs: 1001,1002
      partition 39, leader 1001, replicas: 1002,1001, isrs: 1001,1002
      partition 15, leader 1001, replicas: 1002,1001, isrs: 1001,1002
```
You can also use this output to verify the topics defined by your Kafka cluster, as well as the number of partitions each topic defines. You need this information when copying data between Kafka and Vertica.

Retrieving Messages from a Kafka Topic

When you are troubleshooting issues with streaming messages from Kafka in Vertica, you often want to look at the raw data that Kafka sent. For example, you may want to verify that the messages are in the format that your expect. Or, you may want to review specific messages to see if some of them weren't in the right format for Vertica to parse. You can use kafkacat to read messages from a topic using its consume command (-C). At the very least, you must pass kafkacat the brokers (-b argument) and the topic you want to read from (-t). You can also choose to read messages from a specific offset (-o) and partition (-p). You will usually also want kafkacat to exit after completing the data read (-e) instead continuing to wait for more messages.

This example gets the last message in the topic named web_hits. The offset argument uses a negative value, which tells kafkacat to read from the end of the topic.

```
$ kafkacat -C -b kafka01:6667 -t web_hits -o -1 -e
{"url": "wp-content/list/search.php", "ip": "132.74.240.52",
 "date": "2018/03/28 14:12:34",
 "user_agent": "Mozilla/5.0 (iPod; U; CPU iPhone OS 4_2 like Mac OS X; sl-iSl) AppleWebKit/532.22.4 (KHTML, like Gecko) Version/3.0.5 Mobile/9B117 Safari/6532.22.4")
% Reached end of topic web_hits [0] at offset 54932: exiting
```

You can also read a specific range of messages by specifying an offset and a limit (-c argument). For example, you may want to look at a specific range of data to determine why Vertica could not load it. The following example reads 10 messages from the topic iot-data starting at offset 3280:

```
$ kafkacat -C -b kafka01:6667 -t iot-data -o 3280 -c 10 -e
63680, 19, 24.439323, 26.0128725
43510, 71, 162.319085, -37.4924025
91113, 53, 139.764857, -74.735731
```
Generating Data for a Kafka Topic

You may want to create sample messages if you are preparing to stream data from a Kafka topic that is not yet active. You can then verify that the topic's messages load into Vertica without worrying that you will miss actual data.

To send data to Kafka, use kafkacat's produce command (-P). The easiest way to supply it with messages is to pipe them in via STDIN, one message per line. You can choose a specific partition for the data, or have kafkacat randomly assign each message to a random partition by setting the partition number to -1. For example, suppose you have a file named iot-data.csv that you wanted to produce to random partitions of a Kafka topic named iot-data. Then you could use the following command:

```
$ cat iot_data.csv | kafkacat -P -p -1 -b kafka01:6667 -t iot-data
```
**vkconfig Script Options**

Vertica includes the vkconfig script that lets you configure your schedulers. This script contains multiple tools that set groups of options in the scheduler, as well as starting and shutting it down. You supply the tool you want to use as the first argument in your call to the vkconfig script.

The topics in this section explain each of the tools available in the vkconfig script as well as their options. You can use the options in the Common vkconfig Script Options topic with any of the utilities. Utility-specific options appear in their respective tables.

**Common vkconfig Script Options**

These options are available across the different tools available in the vkconfig script.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--config-schema <em>schema_name</em></td>
<td>The name of the scheduler's Vertica schema. This value is the same as the name of the scheduler. You use this name to identify the scheduler during configuration.</td>
</tr>
<tr>
<td></td>
<td>Default Value: stream_config</td>
</tr>
<tr>
<td>--help</td>
<td>Prints out a help menu listing available options with a description.</td>
</tr>
<tr>
<td>--version</td>
<td>Displays the version number of the scheduler.</td>
</tr>
<tr>
<td>--dbhost <em>host_name</em></td>
<td>The host name or IP address of the Vertica node acting as the initiator node for the scheduler.</td>
</tr>
<tr>
<td></td>
<td>Default Value: localhost</td>
</tr>
<tr>
<td>--dbport <em>port_number</em></td>
<td>The port to use to connect to a Vertica database.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>--username <em>username</em></td>
<td>The Vertica database user used to alter the configuration of the scheduler. This use must have create privileges on the scheduler's schema. <strong>Default Value:</strong> Current user</td>
</tr>
<tr>
<td>--password <em>password</em></td>
<td>Password for the database user.</td>
</tr>
<tr>
<td>--jdbc-url <em>url</em></td>
<td>A complete JDBC URL that vkconfig uses instead of standard JDBC URL string to connect to Vertica.</td>
</tr>
<tr>
<td>--jdbc-opt <em>option=value</em> [&amp;option2=value2...]</td>
<td>One or more options to add to the standard JDBC URL that vkconfig uses to connect to Vertica. Cannot be combined with --jdbc-url.</td>
</tr>
<tr>
<td>--conf <em>filename</em></td>
<td>A text file containing configuration options for the vkconfig script. See Configuration File Format below.</td>
</tr>
<tr>
<td>--create</td>
<td>Creates a new instance of the supplied type.</td>
</tr>
<tr>
<td>--update</td>
<td>Updates an existing instance of the supplied type.</td>
</tr>
<tr>
<td>--delete</td>
<td>Delete an instance of the supplied type.</td>
</tr>
<tr>
<td>--enable-ssl</td>
<td>Enables the vkconfig script to use SSL to connect to Vertica or between Vertica and Kafka. See Configuring Your Scheduler for SSL for more information.</td>
</tr>
<tr>
<td>--ssl-ca-alias <em>alias_name</em></td>
<td>The alias of the root certificate authority in the trust store. Must be set when Vertica uses SSL to connect to...</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Kafka.</td>
<td>The alias of the key and certificate pairs within the key store. Must be set when Vertica uses SSL to connect to Kafka.</td>
</tr>
<tr>
<td>--ssl-key-alias alias_name</td>
<td>The password for the SSL key. Must be set when Vertica uses SSL to connect to Kafka.</td>
</tr>
<tr>
<td>--ssl-key-password password</td>
<td>Caution: Specifying this option on the command line can expose it to other users logged into the host. Always use a configuration file to set this option.</td>
</tr>
</tbody>
</table>

**Configuration File Format**

You can use a configuration file to store common parameters you use in your calls to the `vkconfig` utility. The configuration file is a text file containing one option setting per line in the format:

```
option=value
```

You can also include comments in the option file by prefixing them with a hash mark (#).

```
#config.properties:
username=myuser
password=mypassword
dbhost=localhost
dbport=5433
```

You tell `vkconfig` to use the configuration file using the `--conf` option:

```
$ /opt/vertica/packages/kafka/bin/vkconfig source --update --conf config.properties
```

You can override any stored parameter from the command line:

```
$ /opt/vertica/packages/kafka/bin/vkconfig source --update --conf config.properties --dbhost otherVerticaHost
```
Examples

These examples show how you can use the shared utility options.

Display help for the scheduler utility:

$ /opt/vertica/packages/kafka/bin/vkconfig scheduler --help
This command configures a Scheduler, which can run and load data from configured sources and clusters into Vertica tables. It provides options for changing the 'frame duration' (time given per set of batches to resolve), as well as the dedicated Vertica resource pool the Scheduler will use while running.

Available Options:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>REQUIRED #ARGS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf</td>
<td>no 1</td>
<td>Allow the use of a properties file to associate parameter keys and values. This file enables command string reuse and cleaner command strings.</td>
</tr>
<tr>
<td>help</td>
<td>no 0</td>
<td>Outputs a help context for the given subutility.</td>
</tr>
<tr>
<td>version</td>
<td>no 0</td>
<td>Outputs the current Version of the scheduler.</td>
</tr>
<tr>
<td>skip-validation</td>
<td>no 0</td>
<td>[Depricated] Use --validation-type. Determine what happens when there are configuration errors. Accepts: ERROR - errors out, WARN - prints out a message and continues, SKIP - skip running validations</td>
</tr>
<tr>
<td>validation-type</td>
<td>no 1</td>
<td></td>
</tr>
<tr>
<td>dbhost</td>
<td>no 1</td>
<td>The Vertica database hostname that contains metadata and configuration information. The default value is 'localhost'.</td>
</tr>
<tr>
<td>dbport</td>
<td>no 1</td>
<td>The port at the hostname to connect to the Vertica database. The default value is '5433'.</td>
</tr>
<tr>
<td>username</td>
<td>no 1</td>
<td>The user to connect to HPE Vertica. The default value is the current system user.</td>
</tr>
<tr>
<td>password</td>
<td>no 1</td>
<td>The password for the user connecting to Vertica. The default value is empty.</td>
</tr>
<tr>
<td>jdbc-url</td>
<td>no 1</td>
<td>A JDBC URL that can override Vertica connection parameters and provide additional JDBC options.</td>
</tr>
<tr>
<td>jdbc-opt</td>
<td>no 1</td>
<td>Options to add to the JDBC URL used to connect to Vertica ('&amp;'-separated key=value list). Used with generated URL (i.e. not with '--jdbc-url' set).</td>
</tr>
<tr>
<td>enable-ssl</td>
<td>no 1</td>
<td>Enable SSL between JDBC and Vertica and/or Vertica and Kafka.</td>
</tr>
<tr>
<td>ssl-ca-alias</td>
<td>no 1</td>
<td>The alias of the root CA within the provided trust store used when connecting between Vertica and Kafka.</td>
</tr>
<tr>
<td>ssl-key-alias</td>
<td>no 1</td>
<td>The alias of the key and certificate pair within the provided key store used when connecting between Vertica and Kafka.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Required</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>ssl-key-password</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>config-schema</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>update</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>delete</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>drop</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>operator</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add-operator</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>remove-operator</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>upgrade</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>upgrade-to-schema</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>fix-config</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>frame-duration</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>resource-pool</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>config-refresh</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>new-source-policy</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>eof-timeout-ms</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>pushback-policy</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>pushback-max-count</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>
Scheduler Tool Options

The `vkconfig` script's scheduler tool lets you configure schedulers that continuously loads data from Kafka into Vertica. Use the scheduler tool to create, update, or delete a scheduler, defined by `config-schema`. If you do not specify a scheduler, commands apply to the default `stream_config` scheduler.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--operator username</code></td>
<td>Allows the dbadmin to grant privileges to a previously created Vertica user or role. This option gives the specified user all privileges on the scheduler instance and EXECUTE privileges on the <code>libkafka</code> library and all its UDxs. Granting operator privileges gives the user the right to read data off any source in any cluster that can be reached from the Vertica node. The dbadmin must grant the user separate permission for them to have write privileges on the target tables. Requires the <code>--create</code> shared utility option. Use the <code>--add-operator</code> option to grant operate privileges after the scheduler has been created. To revoke privileges, use the <code>--remove</code> option with the <code>--operator</code> option.</td>
</tr>
<tr>
<td><code>--drop schema_name</code></td>
<td>Drops the specified scheduler's schema. Dropping its schema deletes the scheduler. After you drop the scheduler's schema, you cannot recover it.</td>
</tr>
<tr>
<td><code>--add-operator user_name</code></td>
<td>Grants a Vertica user account or role access to use and alter the scheduler. Requires the <code>--update</code> shared utility option.</td>
</tr>
<tr>
<td><code>--remove-operator user_name</code></td>
<td>Removes access to the scheduler from a Vertica user account. Requires the <code>--update</code> shared utility option.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--upgrade</code></td>
<td>Upgrades the existing scheduler and configuration schema to the current Vertica version. The upgraded version of the scheduler is not backwards compatible with earlier versions. To upgrade a scheduler to an alternate schema, use the <code>upgrade-to-schema</code> parameter. See <a href="#">Updating Schedulers After Vertica Upgrades</a> for more information.</td>
</tr>
<tr>
<td><code>--upgrade-to-schema</code></td>
<td>Copies the scheduler's schema to a new schema specified by <code>schema name</code> and then upgrades it to be compatible with the current version of Vertica. Vertica does not alter the old schema. Requires the <code>--upgrade</code> scheduler utility option.</td>
</tr>
</tbody>
</table>
| `--frame-duration`         | The interval of time that all individual frames last with this scheduler. Vertica must have enough time to complete COPY tasks within this duration. You can approximate the average available time per COPY using the following equation:  
   \[ TimePerCopy = (FrameDuration \times Parallelism) / Microbatches \]  
   This is just a rough estimate as there are many factors that impact the amount of time that each COPY statement will be able to run.  
   Vertica requires at least 100 milliseconds per COPY to function. You can increase the available time per COPY by increasing your frame duration.  
   **Default Value:**  
   00:00:10                                                                                                                   |
| `--config-refresh`         | The interval of time that the scheduler runs before synchronizing its settings and updating its cached metadata (such as changes made by using the `--update` option).  
   **Default Value:**  
   00:05:00                                                                                                                   |
| `--resource-pool`          | The resource pool to be used by all queries executed by this scheduler. You must create this pool in advance if you are not using the default pool.  
   **Default Value:**                                                                                                         |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stream_default_pool</code></td>
<td></td>
</tr>
<tr>
<td>`--new-source-policy {FAIR</td>
<td>START</td>
</tr>
<tr>
<td></td>
<td>- FAIR: Takes the average length of time from the previous batches and schedules itself appropriately.</td>
</tr>
<tr>
<td></td>
<td>- START: All new sources start at the beginning of the frame. The batch receives the minimal amount of time to run.</td>
</tr>
<tr>
<td></td>
<td>- END: All new sources start at the end of the frame. The batch receives the maximum amount of time to run.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> FAIR</td>
</tr>
<tr>
<td><code>--eof-timeout-ms number of milliseconds</code></td>
<td>If a COPY command does not receive any messages within the eof-timeout-ms interval, Vertica responds by ending that COPY statement.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value:</strong> 1 second</td>
</tr>
<tr>
<td></td>
<td>See <a href="https://docs.vertica.com/Documentation/Vertica%20Analytic%20Database%20(9.0.x)/Integrating%20with%20Apache%20Kafka">Manually Copying Data From Kafka</a> for more information.</td>
</tr>
<tr>
<td><code>--message_max_bytes max_message_size</code></td>
<td>The maximum message size, in bytes. <strong>Default Value:</strong> 1048576</td>
</tr>
<tr>
<td><code>--fix-config</code></td>
<td>Repairs the configuration and re-creates any missing tables. Valid only with the --update shared configuration option.</td>
</tr>
<tr>
<td>`--validation-type {ERROR</td>
<td>WARN</td>
</tr>
</tbody>
</table>
### Option Description

- **ERROR** - Cancel configuration or creation if validation fails. If you do not specify a validation type, this value is the default.
- **WARN** - Proceed with task if validation fails, but display a warning.
- **SKIP** - Perform no validation.

For more information on validation, refer to [Automatically Copying Data From Kafka](https://www.vertica.com/docs/Vertica debería ir a la página 5154 del manual del usuario de Vertica).

Renamed from **--skip-validation**.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| **--auto-sync {TRUE|FALSE}** | When TRUE, Vertica automatically synchronizes scheduler source information at the interval specified in **--config-refresh**.  

**Default Value:**  

TRUE  

For more information on synchronization, refer to [Automatically Copying Data From Kafka](https://www.vertica.com/docs/Vertica debería ir a la página 5154 del manual del usuario de Vertica). |

### Examples

These examples show how you can use the scheduler utility options.

Give a user, Jim, privileges on the StreamConfig scheduler. Specify that you are making edits to the stream_config scheduler with the **--config-schema** option:

```
$ /opt/vertica/packages/kafka/bin/vkconfig scheduler --update --config-schema stream_config --add-operator Jim
```

Edit the default stream_config scheduler so that every microbatch waits for data for one second before ending:

```
$ /opt/vertica/packages/kafka/bin/vkconfig scheduler --update --eof-timeout-ms 1000
```

Upgrade the scheduler named iot_scheduler_8.1 to a new scheduler named iot_scheduler_9.0 that is compatible with the current version of Vertica:
Drop the schema scheduler219a:

```bash
$ /opt/vertica/packages/kafka/bin/vkconfig scheduler --drop --config-schema scheduler219a --username dbadmin
```

## Cluster Tool Options

The `vkconfig` script's cluster tool lets you define the streaming hosts your scheduler connects to.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--cluster cluster_name</code></td>
<td>A unique, case insensitive name for the cluster.</td>
</tr>
<tr>
<td><code>--new-cluster cluster_name</code></td>
<td>The updated name for the cluster. Requires the <code>--update</code> shared utility option.</td>
</tr>
<tr>
<td><code>--hosts b1:port[,b2:port...]</code></td>
<td>Identifies the broker hosts that you want to add, edit, or remove from a Kafka cluster. To identify multiple hosts, use a comma delimiter.</td>
</tr>
</tbody>
</table>

### Examples

This example shows how you can create the cluster, StreamCluster1, and assign two hosts:

```bash
$ /opt/vertica/packages/kafka/bin/vkconfig cluster --create --cluster StreamCluster1 \
--hosts 10.10.10.10:9092,10.10.10.11:9092 \
--conf myscheduler.config
```

## Source Tool Options

Use the `vkconfig` script's source tool to create, update, or delete a source.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--source</strong> name</td>
<td>The name for the source to add to, remove from, or edit in the scheduler configuration. You can use any name you like for a new source. Most people use the name of the Kafka topic the scheduler loads its data from.</td>
</tr>
<tr>
<td><strong>--new-source</strong> name</td>
<td>Updates the name of an existing source to the name specified by this parameter.</td>
</tr>
<tr>
<td><strong>--partitions</strong> count</td>
<td>Sets the number of partitions in the source.</td>
</tr>
<tr>
<td><strong>--cluster</strong> name</td>
<td>Identifies the cluster containing the source that you want to create or edit. You must have already defined this cluster in the scheduler.</td>
</tr>
<tr>
<td><strong>--new-cluster</strong> name</td>
<td>Changes the cluster this source belongs to. All sources referencing the old cluster source now target this cluster.</td>
</tr>
<tr>
<td><strong>--enabled</strong> TRUE</td>
<td>When TRUE, the source is available for use.</td>
</tr>
</tbody>
</table>

**Examples**

The following examples show how you can create or update SourceFeed.

Create the source SourceFeed and assign it to the cluster, StreamCluster1 in the scheduler defined by the myscheduler.conf config file:

```
$ /opt/vertica/packages/kafka/bin/vkconfig source --create --source SourceFeed \   
   --cluster StreamCluster1 --partitions 3 \   
   --conf myscheduler.conf
```
Update the existing source SourceFeed to use the existing cluster, StreamCluster2 in the scheduler defined by the myscheduler.conf config file:

```
$ /opt/vertica/packages/kafka/bin/vkconfig source --update --source SourceFeed
   --new-cluster StreamCluster2
   --conf myscheduler.conf
```

### Target Tool Options

Use the target tool to configure a Vertica table to receive data from your streaming data application.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--target-schema</code></td>
<td>The existing Vertica target schema associated with this target.</td>
</tr>
<tr>
<td><code>schema.table_name</code></td>
<td></td>
</tr>
<tr>
<td><code>--new-target-schema</code></td>
<td>Changes the Vertica schema associated with this schema to a new, already created schema.</td>
</tr>
<tr>
<td><code>schema_name</code></td>
<td></td>
</tr>
<tr>
<td><code>--target-table</code></td>
<td>The name of a Vertica table corresponding to the target.</td>
</tr>
<tr>
<td><code>table_name</code></td>
<td></td>
</tr>
<tr>
<td><code>--new-target-table</code></td>
<td>Changes the Vertica target table associated with this schema to a new, already created table.</td>
</tr>
<tr>
<td><code>schema_name</code></td>
<td></td>
</tr>
</tbody>
</table>

**Requires:** `--update` option.

**Important:** Avoid having columns with primary key restrictions in your target table. The scheduler stops loading data if it encounters a row that has a value which violates this restriction. If you must have a primary key restricted column, try to filter out any redundant values for that column in the streamed data before it is loaded by the scheduler.

### Examples

This example shows how you can create a target for the scheduler defined in the myscheduler.conf configuration file from public.streamtarget table:

```
$ /opt/vertica/packages/kafka/bin/vkconfig target --create --target-table streamtarget --conf myscheduler.conf
```
# Load Spec Tool Options

The vkconfig script's load spec tool lets you provide parameters for the COPY statement that loads streaming data.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--load-spec <em>spec_name</em></td>
<td>A unique name for this copy load spec.</td>
</tr>
<tr>
<td>--filters &quot;<em>filter_name</em>&quot;</td>
<td>A Vertica FILTER chain containing all UDFilters for the COPY statement. For more information on filters, refer to Parsing Custom Formats.</td>
</tr>
<tr>
<td>--message-max-bytes <em>max_message_size</em></td>
<td>Specifies the maximum size, in bytes, of a Kafka message. Default Value: 1048576</td>
</tr>
<tr>
<td>--parser <em>parser name</em></td>
<td>Identifies a Vertica UDParser to use with a specified target. This parser is used within the COPY statement that the scheduler runs to load data. If you are using a Vertica native parser, the values supplied to the --parser-parameters option are passed through to the COPY statement. Default Value: KafkaParser</td>
</tr>
<tr>
<td>--parser-parameters &quot;<em>key=value, key=value</em>&quot;</td>
<td>A list of parameters to provide to the parser specified in the --parser parameter. When you use a Vertica native parser, the scheduler passes these parameters to the COPY statement where they are in turn passed to the parser.</td>
</tr>
<tr>
<td>--new-load-spec <em>new_name</em></td>
<td>A new, unique name for an existing load spec. Requires the --update parameter.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--uds-kv-parameters key=value, key=value</td>
<td>A comma separated list of key value pairs for the user-defined source.</td>
</tr>
<tr>
<td>--load-method AUTO</td>
<td>TRICKLE</td>
</tr>
</tbody>
</table>

**Examples**

These examples show how you can use the Load Spec utility options.

Create the load spec, Streamspec1:

```
$ /opt/vertica/packages/kafka/bin/vkconfig load-spec --create --load-spec Streamspec1 --conf myscheduler.conf
```

Update the load spec, Streamspec1, to the name Streamspec2, and also update the load method to Direct:

```
$ /opt/vertica/packages/kafka/bin/vkconfig load-spec --update --load-spec Streamspec1 \
direct \  
--new-load-spec Streamspec2 --load-method \  
--conf myscheduler.conf
```

Update the load spec, Filterspec, to use the KafkaInsertLengths filter and a custom decryption filter:

```
$ /opt/vertica/packages/kafka/bin/vkconfig load-spec --update --load-spec Filterspec \ 
(parameter=Key)" \
--filters "KafkaInsertLengths() DecryptFilter \ 
--conf myscheduler.conf
```

**Microbatch Tool Options**

The vkconfig script's microbatch tool lets you configure a scheduler's microbatches.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--microbatch name</strong></td>
<td>A unique, case insensitive name for the microbatch.</td>
</tr>
<tr>
<td><strong>--new-microbatch updated_name</strong></td>
<td>The updated name for the microbatch. Requires the <strong>--update</strong> shared utility option.</td>
</tr>
<tr>
<td><strong>--load-spec loadspec_name</strong></td>
<td>The load spec to use while processing this microbatch.</td>
</tr>
<tr>
<td><strong>--target-schema schema_name</strong></td>
<td>The existing Vertica target schema associated with this microbatch.</td>
</tr>
<tr>
<td><strong>--rejection-schema schema_name</strong></td>
<td>The existing Vertica schema that contains a table for storing rejected messages.</td>
</tr>
</tbody>
</table>
| **--target-columns column_name, ...|column expression**                                                          | A column expression for the target table. This value can be a list of columns or a complete expression.  
| | **--rejection-table table_name**                             | The existing Vertica table that stores rejected messages.                                                                                   |
| **--enabled TRUE|FALSE**                                                        | When TRUE, allows the microbatch to execute.                                                                                                  |
| **--target-table table_name**                                | The name of a Vertica table corresponding to the target. This table must belong to the target schema.                                         |
| **--add-source source_name**                                 | The name of a source to assign to this microbatch. You can use this parameter once per command. You can also use it with **--update** to add sources to a microbatch. Requires **--add-source-cluster**. |
| **--add-source-cluster cluster_name**                        | The name of a cluster to assign to this microbatch. You can use this parameter once per command. You can also use it with **--update** to add sources to a microbatch. You can only add sources from the same cluster to a single microbatch. Requires **--add-source**. |
Option | Description
--- | ---
--remove-source source_name | The name of a source to remove from this microbatch. You can use this parameter once per command. You can also use it with --update to remove multiple sources from a microbatch. Requires --remove-source-cluster.

--remove-source-cluster cluster_name | The name of a cluster to remove from this microbatch. You can use this parameter once per command. Requires --remove-source.

--offset partition_1_offset [,partition_2_offset,...] | The offset of the message in the source where the microbatch starts its load. If you use this parameter, you must supply an offset value for each partition in the source or each partition you list in the --partition option.

You can use this option to skip some messages in the source or reload previously read messages.

**Important:** You cannot set an offset for a microbatch while the scheduler is running. If you attempt to do so, the vkconfig utility returns an error. Use the shutdown utility to shut the scheduler down before setting an offset for a microbatch.

--partition partition_1 [,partition_2,...] | One or more partitions to which the offsets given in the --offset option apply. If you supply this option, then the offset values given in the --offset option applies to the partitions you specify. Requires the --offset option.

--source source_name | The name of the source to which the offset in the --offset option applies. Required when the microbatch defines more than one source or the --cluster parameter is given. Requires the --offset option.

--cluster cluster_name | The name of the cluster to which the --offset option applies. Only required if the microbatch defines more than one cluster or the --source parameter is supplied. Requires the --offset option.

### Examples

This example shows how you can create the microbatch, mbatch1. This microbatch identifies the schema, target table, load spec, and source for the microbatch:
Launch Tool Options

Use the vkconfig script's launch tool to assign a name to a scheduler instance.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--enable-ssl {true</td>
<td>false}</td>
</tr>
<tr>
<td>--ssl-ca-alias alias</td>
<td>The user-defined alias of the root certifying authority you are using to authenticate communication between Vertica and Kafka. This parameter is used only when SSL is enabled.</td>
</tr>
<tr>
<td>--ssl-key-alias alias</td>
<td>The user-defined alias of the key/certificate pair you are using to authenticate communication between Vertica and Kafka. This parameter is used only when SSL is enabled.</td>
</tr>
<tr>
<td>--ssl-key-password password</td>
<td>The password used to create your SSL key. This parameter is used only</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--instance-name name</td>
<td>(Optional) Allows you to name the process running the scheduler. You can use this command when viewing the scheduler_history table, to find which instance is currently running.</td>
</tr>
</tbody>
</table>

## Examples

This example shows how you can launch the scheduler defined in the myscheduler.conf config file and give it the instance name PrimaryScheduler:

```
$ /opt/vertica/packages/kafka/bin/vkconfig launch --instance-name PrimaryScheduler --conf myscheduler.conf
```

This example shows how you can launch an instance named SecureScheduler with SSL enabled:

```
$ /opt/vertica/packages/kafka/bin/vkconfig launch --instance-name SecureScheduler --enable-SSL true \
   --ssl-ca-alias authenticcert --ssl-key-alias ourkey \
   --ssl-key-password secret \
   --conf myscheduler.conf
```

## Shutdown Tool Options

Use the vkconfig script’s shutdown tool to terminate all Vertica schedulers running on a host. Always run this command before restarting a scheduler to ensure the scheduler has shutdown correctly.

**Note:** Launching a scheduler without terminating any existing instances of it can produce unexpected behavior.
Examples

This example shows how you can terminate all Vertica Kafka schedulers:

```bash
$ /opt/vertica/packages/kafka/bin/vkconfig shutdown --conf myscheduler.conf
```
Kafka Function Reference

This section lists the functions that make up Vertica's Kafka integration feature.

KafkaAvroParser

The KafkaAvroParser parses Avro-formatted Kafka messages and loads them into a regular Vertica table or a Vertica flex table.

KafkaAvroParser(
    [reject_on_materialized_type_error=Boolean]
    [, flatten_maps=Boolean]
    [, flatten_arrays=Boolean]
    [, flatten_records=Boolean]
    [, external_schema=JSON_string]
    [, codec='default''snappy''null']
    [, with_metadata=Boolean]
    [, schema-registry-url='url']
    [, schema_registry_subject='subject_name']
    [, schema_registry_version='version_number']
)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject_on_materialized_type_error</td>
<td>When set to TRUE, rejects the data row if it contains a materialized column value that cannot be mapped into the materialized column's data type.</td>
</tr>
<tr>
<td>flatten_maps</td>
<td>If set to TRUE, flattens all Avro maps.</td>
</tr>
<tr>
<td>flatten_arrays</td>
<td>If set to TRUE, flattens Avro arrays.</td>
</tr>
<tr>
<td>flatten_records</td>
<td>If set to TRUE, flattens all Avro records.</td>
</tr>
<tr>
<td>external_schema</td>
<td>Specifies the schema of the Avro file as a JSON string. If this parameter is not specified, the parser assumes that each message has the schema on it. If you are using a schema registry, do not use this parameter.</td>
</tr>
<tr>
<td>codec</td>
<td>Specifies the codec in which the Avro file was written. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>* 'default' - Avro's default</td>
</tr>
<tr>
<td></td>
<td>* 'snappy' - snappy compression</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>'null'</td>
<td>data is not compressed and codec is not needed</td>
</tr>
<tr>
<td></td>
<td>Note: This option is mainly provided for backwards compatibility. You usually have Kafka compress data at the message level, and have KafkaSource decompress the message for you.</td>
</tr>
<tr>
<td>with_metadata</td>
<td>If set to TRUE, messages include Avro datum, schema, and object metadata. By default, the KafkaAvroParser parses messages without including schema and metadata. If you enable this parameter, write your messages using the Avro API and confirm they contain only Avro datum. The default value is FALSE.</td>
</tr>
<tr>
<td>schema_registry_url</td>
<td>Specifies the URL of the Confluent schema registry. This parameter is required to load data based on a schema registry version. If you are using an external schema, do not use this parameter. For more information, refer to Using a Schema Registry with Kafka.</td>
</tr>
<tr>
<td>schema_registry_subject</td>
<td>In the schema registry, the subject of the schema to use for data loading.</td>
</tr>
<tr>
<td>schema_registry_version</td>
<td>In the schema registry, the version of the schema to use for data loading.</td>
</tr>
</tbody>
</table>

See Loading Avro Data for more information.

The following example demonstrates loading data from Kafka in an Avro format. The statement:

- Loads data into an existing flex table named weather_logs.
- Copies data from the default Kafka broker (running on the local system on port 9092).
- The source is named temperature.
- The source has a single partition.
- The load starts from offset 0.
The load ends either after 10 seconds or the load reaches the end of the source, whichever occurs first.

The KafkaAvroParser does not flatten any arrays, maps, or records it finds in the source.

The schema for the data is provided in the statement as a JSON string. It defines a record type named Weather that contains fields for a station name, a time, and a temperature.

Rejected rows of data are saved to a table named t_rejects1.

```sql
=> COPY weather_logs
    SOURCE KafkaSource(stream='temperature|0|0', stop_on_eof=true,
                        duration=interval '10 seconds')
    PARSER KafkaAvroParser(flatten_arrays=False, flatten_maps=False, flatten_records=False,
                            external_schema=E'{"type":"record","name":"Weather","fields":
                            
                            [{"name":"station","type":"string"},
                             {"name":"time","type":"long"},
                             {"name":"temp","type":"int"}]}')
    REJECTED DATA AS TABLE "t_rejects1";
```

**KafkaExport**

Sends Vertica data to Kafka.

If Vertica successfully exports all of the rows of data to Kafka, this function returns zero rows. You can use the output of this function to copy failed messages to a secondary table for evaluation and reprocessing.

**Syntax**

```
SELECT KafkaExport(partitionColumn, keyColumn, valueColumn
                    USING PARAMETERS brokers='host[:port][,host...]',
                    topic='topicname'
                    [,kafka_conf='kafka_configuration_setting']
                    OVER (partition_clause) FROM table;
```

**Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>partitionColumn</code></td>
<td>The target partition for the export. If you set this value to NULL, Vertica uses the default partitioning scheme.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>You can use the partition argument to send messages to partitions that map to Vertica segments.</td>
<td></td>
</tr>
<tr>
<td>The user defined key value associated with the valueColumn. Use NULL to skip this argument.</td>
<td></td>
</tr>
<tr>
<td>The message itself. The column is a LONG VARCHAR, allowing you to send up to 32MB of data to Kafka. However, Kafka may impose its own limits on message size.</td>
<td></td>
</tr>
<tr>
<td>A string containing a comma-separated list of one or more host names or IP addresses (with optional port number) of brokers in the Kafka cluster.</td>
<td></td>
</tr>
<tr>
<td>The Kafka topic to which you are exporting.</td>
<td></td>
</tr>
<tr>
<td>A string containing one or more rdkafka options in the form:</td>
<td></td>
</tr>
<tr>
<td>Vertica passes these options to rdkafka library, which it uses to communicate with Kafka. Separate multiple settings using semicolons. See Vertica Producer Settings for the settings you can use.</td>
<td></td>
</tr>
</tbody>
</table>

Examples

```sql
=> SELECT KafkaExport(partition, messageId, message
                       USING PARAMETERS brokers='kafka01.example.com:9092',
                       source='failure_test',
                       kafka_conf='message.max.bytes=64000')
OVER (PARTITION BEST)
FROM failure_test;

<table>
<thead>
<tr>
<th>partition</th>
<th>key</th>
<th>substr</th>
<th>failure_reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>key1</td>
<td>negative partition not allowed</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>54321</td>
<td></td>
<td>nonexistant partition</td>
<td>Local: Unknown partition</td>
</tr>
<tr>
<td>0</td>
<td>normal key1</td>
<td>normal value1</td>
<td>Broker: Message size too</td>
</tr>
<tr>
<td>.large</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.large</td>
<td>normal key2</td>
<td>normal value2</td>
<td>Broker: Message size too</td>
</tr>
<tr>
<td>.large</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See Also

Producing Data Using KafkaExport

**KafkaJSONParser**

The KafkaJSONParser parses JSON-formatted Kafka messages and loads them into a regular Vertica table or a Vertica flex table.

The syntax for calling the parser is:

```java
KafkaJSONParser(
    [flatten_maps=Boolean]
    [, flatten_arrays=Boolean]
    [, start_point=Boolean]
    [, omit_empty_keys=Boolean]
    [, reject_on_duplicate=Boolean]
    [, reject_on_materialized_type_error=Boolean]
    [, reject_on_empty_key=Boolean])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_maps</td>
<td>Flattens all JSON maps if set to TRUE</td>
</tr>
<tr>
<td>flatten_arrays</td>
<td>Flattens JSON arrays if set to TRUE</td>
</tr>
<tr>
<td>start_point</td>
<td>Specifies the name of a key in the JSON load data at which to begin parsing. The parser ignores all data before the start_point value. The parser processes data after the first instance, and up to the second, ignoring any remaining data.</td>
</tr>
<tr>
<td>omit_empty_keys</td>
<td>If set to TRUE, omits any key from the load data that does not have a value set.</td>
</tr>
<tr>
<td>reject_on_duplicate</td>
<td>If set to TRUE, rejects data that contains duplicate key names.</td>
</tr>
<tr>
<td>reject_on_materialized_type_error</td>
<td>When set to TRUE, rejects the data row if the data includes keys matching an existing materialized column and has a key that cannot be mapped into the materialized column's data type.</td>
</tr>
<tr>
<td>reject_on_empty_key</td>
<td>If set to TRUE, rejects any row containing a key without a value.</td>
</tr>
</tbody>
</table>
See Loading JSON Data for more information.

The following example demonstrates loading JSON data from Kafka. The parameters in the statement define to the load to:

- Load data into the pre-existing table named logs.
- The KafkaSource streams the data from a single partition in the source called server_log.
- The Kafka broker for the data load is running on the host named kafka01 on port 9092.
- KafkaSource stops loading data after either 10 seconds or on reaching the end of the stream, whichever happens first.
- The KafkaJSONParser flattens any arrays or maps in the JSON data.

```
=> COPY logs SOURCE KafkaSource(stream='server_log|0|0',
    stop_on_eof=true,
    duration=interval '10 seconds',
    brokers='kafka01:9092')
PARSER KafkaJSONParser(flatten_arrays=True, flatten_maps=True);
```

## KafkaOffsets

The KafkaOffsets user-defined transform function returns load operation statistics generated by the most recent invocation of KafkaSource. Query KafkaOffsets to see the metadata produced by your most recent load operation. You can query KafkaOffsets after each KafkaSource invocation to view information about that load. If you are using the scheduler, you can also view historical load information in the `stream_microbatch_history` table.

For each load operation, KafkaOffsets returns the following:

- source kafka topic
- source kafka partition
- starting offset
- ending offset
- number of messages loaded
- number of bytes read
- duration of the load operation
The following example demonstrates calling KafkaOffsets to show partition information on the table named web_test that was loaded using KafkaSource.

```
=> SELECT kpartition, start_offset, end_offset, msg_count, ending FROM (select KafkaOffsets() over() FROM web_test) AS stats ORDER BY kpartition;
```

<table>
<thead>
<tr>
<th>kpartition</th>
<th>start_offset</th>
<th>end_offset</th>
<th>msg_count</th>
<th>ending</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-2</td>
<td>9999</td>
<td>1068</td>
<td>END_OFFSET</td>
</tr>
</tbody>
</table>

The output shows that KafkaSource loaded 1068 messages (rows) from Kafka in a single partition. The KafkaSource ended the data load because it reached the ending offset.

**Note:** The values shown in the start_offset column are exclusive (the message with the shown offset was not loaded) and the values in the end_offset column are inclusive (the message with the shown offset was loaded). This is the opposite of the values specified in the KafkaSource's stream parameter. The difference between the inclusiveness of KafkaSource's and KafkaOffset's start and end offsets are based on the needs of the job scheduler. KafkaOffset is primarily intended for the job scheduler's use, so the start and end offset values are defined so the scheduler can easily start streaming from where left off.

**KafkaParser**

The KafkaParser does not parse data loaded from Kafka. Instead, it passes the messages through as LONG VARCHAR values. Use this parser when you want to load raw Kafka messages into Vertica for further processing. You can use this parser as a catch-all for unsupported formats.

KafkaParser does not take any parameters.

**Example**

The following example loads raw messages from a Kafka topic named iot-data into a table named raw_iot.

```
=> CREATE TABLE raw_iot(message LONG VARCHAR);
```
KafkaSource

The KafkaSource UDL accesses data from a Kafka cluster. All Kafka parsers must use KafkaSource. Messages processed by KafkaSource must be at least one byte in length. KafkaSource writes an error message to vertica.log for zero length messages.

The syntax for calling KafkaSource is:

```sql
KafkaSource(
    stream='topic_name|partition|start_offset[|end_offset][,...]',
    [ , brokers='host:port[,...]' ]
    [ , duration=interval ]
    [ , executionparallelism='value' ]
    [ , stop_on_eof=Boolean ]
    [ , eof_timeout=timeout ]
)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Required. Defines the data to be loaded as a comma-separated list of one or more partitions. Each partition is defined by three required values and one optional value separated by pipe characters (</td>
</tr>
<tr>
<td></td>
<td>- topic_name: the name of the Kafka topic to</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>load</strong></td>
<td>load data from. You can read from different Kafka topics in the same stream parameter, with some limitations. See <a href="#">Loading from Multiple Topics in the Same Stream Parameter</a> below for more information.</td>
</tr>
<tr>
<td></td>
<td>- <strong>partition</strong>: the partition in the Kafka topic to copy.</td>
</tr>
<tr>
<td></td>
<td>- <strong>start_offset</strong>: the offset in the Kafka topic where the load will begin. This offset is inclusive (the message with the offset <strong>start_offset</strong> is loaded).</td>
</tr>
<tr>
<td></td>
<td>- <strong>end_offset</strong>: the optional offset where the load should end. This offset is exclusive (the message with the offset <strong>end_offset</strong> will not be loaded). To end a load using <strong>end_offset</strong>, you must supply an ending offset value for all partitions in the stream parameter. Attempting to set an ending offset for some partitions and not and not set offset values for others results in an error.</td>
</tr>
<tr>
<td></td>
<td>- If you do not specify an ending offset, you must supply at least one other ending condition using <strong>stop_on_eof</strong> or <strong>duration</strong>.</td>
</tr>
<tr>
<td><strong>brokers</strong></td>
<td>A comma-separated list of host:port pairs of the brokers in the Kafka cluster. Vertica recommends running Kafka on a different machine than Vertica.</td>
</tr>
<tr>
<td></td>
<td><strong>Default Value</strong>: localhost:9092</td>
</tr>
<tr>
<td><strong>duration</strong></td>
<td>An INTERVAL that specifies the duration of the frame. After this specified amount of time, KafkaSource terminates the COPY statements. If this parameter is not set, you must set at least one other ending condition by using <strong>stop_on_eof</strong> or specify an ending offset instead. See <a href="#">Duration Note</a> below for more information.</td>
</tr>
</tbody>
</table>
Loading from Multiple Topics in the Same Stream Parameter

You can load from multiple Kafka topics in a single stream parameter as long as you follow these guidelines:

- The data for the topics must be in the same format because you pass the data from KafkaSource to a single parser. For example, you cannot load data from one topic that is in
Avro format and another in JSON format.

- Similarly, you need to be careful if you are loading Avro data and specifying an external schema from a registry. The Avro parser accepts a single schema per data load. If the data from the separate topics have different schemas, then all of the data from one of the topics will be rejected.

- The data in the different topics should have the same (or very similar) schemas, especially if you are loading data into a traditional Vertica table. While you can load data with different schemas into a flex table, there are only a few scenarios where it makes sense to combine dissimilar data into a single table.

**Duration Note**

The duration parameter applies to the length of time that Vertica allows the KafkaSource function to run. It usually reflects the amount of time the overall load statement takes. However, if KafkaSource is loading a large volume of data or the data needs extensive processing and parsing, the overall runtime of the query can exceed the amount of time specified in duration.

**Example**

The following example demonstrates calling KafkaSource to load data from Kafka into an existing flex table named web_test with the following options:

- The stream is named web_hits which has a single partition.

- The load starts at the earliest message in the stream (identified by passing -2 as the start offset).

- The load ends when it reaches the message with offset 10000.

- The Kafka cluster’s brokers are kafka01 and kafka03 in the example.com domain.

- The brokers are listening on port 6667. (This is port number that the Hortonworks Hadoop distribution assigns to its Kafka brokers.)

- The load ends if it reaches the end of the stream before reaching the message with offset 10000. If you do not supply this option, the connector waits until Kafka sends a message
with offset 10000.

- The loaded data is sent to the KafkaJSONParser for processing.

```sql
=> COPY web_test
    SOURCE KafkaSource(stream='web_hits|0|-2|10000',
                        brokers='kafka01.example.com:6667,kafka03.example.com:6667',
                        stop_on_eof=true)
    PARSER KafkaJSONParser();
Rows Loaded
----------
 1068
(1 row)
```
Data Streaming Schema Tables

Every time you create a scheduler (--create), Vertica creates a schema for that scheduler with the name you specify or the default stream_config. Each schema has the following tables:

- stream_clusters
- stream_events
- stream_load_specs
- stream_microbatch_history
- stream_microbatch_source_map
- stream_microbatches
- stream_scheduler
- stream_scheduler_history
- stream_sources
- stream_targets

Caution: Vertica recommends that you do not alter these tables except in consultation with support.

stream_clusters

This table lists clusters and hosts. You change settings in this table using the vkconfig cluster tool. See Cluster Tool Options for more information.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>The identification number assigned to the cluster.</td>
</tr>
<tr>
<td>cluster</td>
<td>VARCHAR</td>
<td>The name of the cluster.</td>
</tr>
<tr>
<td>hosts</td>
<td>VARCHAR</td>
<td>A comma-separated list of hosts associated with the cluster.</td>
</tr>
</tbody>
</table>
Examples

This example shows a cluster and its associated hosts.

```sql
=> SELECT * FROM stream_config.stream_clusters;
```

<table>
<thead>
<tr>
<th>id</th>
<th>cluster</th>
<th>hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2250001</td>
<td>streamcluster</td>
<td>10.10.10.10:9092,10.10.10.11:9092</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1 rows)</td>
</tr>
</tbody>
</table>

**stream_events**

This table logs microbatches and other important events from the scheduler in an internal log table.

This table was renamed from kafka_config.kafka_events.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_time</td>
<td>TIMESTAMP</td>
<td>The time the event was logged.</td>
</tr>
<tr>
<td>log_level</td>
<td>VARCHAR</td>
<td>The type of event that was logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid Values:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TRACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DEBUG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FATAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• WARN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• INFO</td>
</tr>
<tr>
<td>frame_start</td>
<td>TIMESTAMP</td>
<td>The time when the frame executed.</td>
</tr>
<tr>
<td>frame_end</td>
<td>TIMESTAMP</td>
<td>The time when the frame completed.</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>microbatch</td>
<td>INTEGER</td>
<td>The identification number of the associated microbatch.</td>
</tr>
<tr>
<td>message</td>
<td>VARCHAR</td>
<td>A description of the event.</td>
</tr>
<tr>
<td>exception</td>
<td>VARCHAR</td>
<td>If this log is in the form of a stack trace, this column lists the exception.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows typical rows from the `stream_events` table.

```sql
=> SELECT * FROM stream_config.stream_events;
-[ RECORD 1 ]-+------------------------+
  event_time | 2016-07-17 13:20:35.548-04
  log_level  | INFO
  frame_start| 2015-07-17 12:28:45.633
  frame_end  | 2015-07-17 13:28:50.701-04
  microbatch |
  message    | New leader registered for schema stream_config. New ID: 0, new Host: 10.20.100.62
              | 004:9092,eng-g9-005:9092), resource pool: kafka_default_pool
  exception  |
-[ RECORD 2 ]-+------------------------+
  event_time | 2016-07-17 13:28:45.643-04
  log_level  | INFO
  frame_start| 2015-07-17 12:28:45.633
  frame_end  | 2015-07-17 13:28:50.701-04
  microbatch |
  message    | Generated tuples: test3|2|-2,test3|1|-2,test3|0|-2
  exception  |
-[ RECORD 3 ]-+------------------------+
  event_time | 2016-07-17 14:28:50.701-04
  log_level  | INFO
  frame_start| 2016-07-17 13:28:45.633
  frame_end  | 2016-07-17 14:28:50.701-04
  microbatch |
  message    | Total rows inserted: 0
  exception  |
```

**stream_load_specs**

This table describes user-created load specs. You change the entries in this table using the `vkconfig` utility's `load spec tool`. 
<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>The identification number assigned to the cluster.</td>
</tr>
<tr>
<td>load_spec</td>
<td>VARCHAR</td>
<td>The name of the load spec.</td>
</tr>
<tr>
<td>filters</td>
<td>VARCHAR</td>
<td>A comma-separated list of hosts associated with the cluster.</td>
</tr>
<tr>
<td>parser</td>
<td>VARCHAR</td>
<td>A VerticaUDParser to use with a specified target. If you are using a Vertica native parser, parser parameters serve as a COPY statement parameters.</td>
</tr>
<tr>
<td>parser_parameters</td>
<td>VARCHAR</td>
<td>A list of parameters to provide to the parser.</td>
</tr>
<tr>
<td>load_method</td>
<td>VARCHAR</td>
<td>The COPY load method to use for all loads with this scheduler. See the COPY statement for more information.</td>
</tr>
<tr>
<td>message_max_bytes</td>
<td>INTEGER</td>
<td>The maximum size, in bytes, of a message.</td>
</tr>
<tr>
<td>uds_kv_parameters</td>
<td>VARCHAR</td>
<td>A list of parameters that are supplied to the KafkaSource statement. If the value in this column is in the format key=value, the scheduler it to the COPY statement's KafkaSource call.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows the load specs that you can use with a Vertica instance.

```
SELECT * FROM stream_config.stream_load_specs;
-| RECORD 1 |---------------|
id       | 1             |
load_spec | loadspec2     |
filters   |               |
parser    | KafkaParser   |
parser_parameters |         |
load_method | direct       |
message_max_bytes | 1048576 |
uds_kv_parameters |         |
-| RECORD 2 |---------------|
id       | 750001        |
load_spec | streamspec1   |
filters   |               |
parser    | KafkaParser   |
parser_parameters |         |
load_method | TRICKLE      |
```
stream_microbatch_history

This table contains a history of every microbatch executed within this scheduler configuration.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_name</td>
<td>VARCHAR</td>
<td>The name of the source.</td>
</tr>
<tr>
<td>source_cluster</td>
<td>VARCHAR</td>
<td>The name of the source cluster. The clusters are defined in stream_clusters.</td>
</tr>
<tr>
<td>source_partition</td>
<td>INTEGER</td>
<td>The number of the data streaming partition.</td>
</tr>
<tr>
<td>start_offset</td>
<td>INTEGER</td>
<td>The starting offset of the microbatch.</td>
</tr>
<tr>
<td>end_offset</td>
<td>INTEGER</td>
<td>The ending offset of the microbatch.</td>
</tr>
<tr>
<td>end_reason</td>
<td>VARCHAR</td>
<td>An explanation for why the batch ended. The following are valid end reasons:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DEADLINE - The batch ran out of time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• END_OFFSET - The load reached the ending offset specified in the KafkaSource. This reason is never used by the scheduler, as it does specify an end offset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• END_OF_STREAM - There are no messages available to the scheduler or the eof_timeout has been reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NETWORK_ERROR - The scheduler could not connect to Kafka</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RESET_OFFSET - The start offset was changed using the --update and --offset parameters to the KafkaSource. This state does not occur during normal scheduler operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SOURCE_ERROR - The specified Kafka topic does not</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exist</td>
<td></td>
<td>The batch ended for an unknown reason</td>
</tr>
<tr>
<td>end_reason_message</td>
<td>VARCHAR</td>
<td>If the end reason is a network or source issue, this column contains a brief description of the issue.</td>
</tr>
<tr>
<td>partition_bytes</td>
<td>INTEGER</td>
<td>The number of bytes transferred from a source partition to a Vertica target table.</td>
</tr>
<tr>
<td>partition_messages</td>
<td>INTEGER</td>
<td>The number of messages transferred from a source partition to a Vertica target table.</td>
</tr>
<tr>
<td>microbatch_id</td>
<td>INTEGER</td>
<td>The Vertica transaction id for the batch session.</td>
</tr>
<tr>
<td>microbatch</td>
<td>VARCHAR</td>
<td>The name of the microbatch.</td>
</tr>
<tr>
<td>target_schema</td>
<td>VARCHAR</td>
<td>The name of the target schema.</td>
</tr>
<tr>
<td>target_table</td>
<td>VARCHAR</td>
<td>The name of the target table.</td>
</tr>
<tr>
<td>timeslice</td>
<td>INTERVAL</td>
<td>The amount of time spent in the KafkaSource operator.</td>
</tr>
<tr>
<td>batch_start</td>
<td>TIMESTAMP</td>
<td>The time the batch executed.</td>
</tr>
<tr>
<td>batch_end</td>
<td>TIMESTAMP</td>
<td>The time the batch completed.</td>
</tr>
<tr>
<td>last_batch_duration</td>
<td>INTERVAL</td>
<td>The length of time required to run the complete COPY statement.</td>
</tr>
<tr>
<td>consecutive_error_count</td>
<td>INTEGER</td>
<td>(Currently not used.) The number of times a microbatch has encountered an error on an attempt to load. This value increases over multiple attempts.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>INTEGER</td>
<td>The identifier for the transaction within the session.</td>
</tr>
<tr>
<td>frame_start</td>
<td>TIMESTAMP</td>
<td>The time the frame started. A frame can contain multiple microbatches.</td>
</tr>
<tr>
<td>frame_end</td>
<td>TIMESTAMP</td>
<td>The time the frame completed.</td>
</tr>
</tbody>
</table>
Examples

This example shows typical rows from the stream_microbatch_history table.

```sql
=> SELECT * FROM stream_config.stream_microbatch_history;

- [ RECORD 1 ]----------------------------
source_name | streamsource1
source_cluster | kafka-1
source_partition | 0
start_offset | 196
end_offset | 196
end_reason | END_OF_STREAM
partition_bytes | 0
partition_messages | 0
microbatch_id | 1
microbatch | mb_0
target_schema | public
target_table | kafka_flex_0	
timeslice | 00:00:00.892
batch_start | 2016-07-28 11:31:25.854221
batch_end | 2016-07-28 11:31:26.357942
last_batch_duration | 00:00:00.379826
consecutive_error_count | 
transaction_id | 45035996275130064
frame_start | 2016-07-28 11:31:25.751
frame_end | 
end_reason_message | 

- [ RECORD 2 ]----------------------------
source_name | streamsource1
source_cluster | kafka-1
source_partition | 1
start_offset | 197
end_offset | 197
end_reason | NETWORK_ISSUE
partition_bytes | 0
partition_messages | 0
microbatch_id | 1
microbatch | mb_0
target_schema | public
target_table | kafka_flex_0	
timeslice | 00:00:00.897
batch_start | 2016-07-28 11:31:45.84898
batch_end | 2016-07-28 11:31:46.253367
last_batch_duration | 00:00:00.377796
consecutive_error_count | 
transaction_id | 45035996275130109
frame_start | 2016-07-28 11:31:45.751
frame_end | 
end_reason_message | Local: All brokers are down
stream_microbatch_source_map

This table maps microbatches to their associated sources.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microbatch</td>
<td>INTEGER</td>
<td>The identification number of the microbatch.</td>
</tr>
<tr>
<td>source</td>
<td>INTEGER</td>
<td>The identification number of the associated source.</td>
</tr>
</tbody>
</table>

Examples

This example shows typical rows from the stream_microbatch table.

```
SELECT * FROM stream_config.stream_microbatch_source_map;
```

```
+----------+-------+
<table>
<thead>
<tr>
<th>microbatch</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
+----------+-------+
(2 rows)
```

stream_microbatches

This table contains configuration data related to microbatches.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>The identification number of the microbatch.</td>
</tr>
<tr>
<td>microbatch</td>
<td>VARCHAR</td>
<td>The name of the microbatch.</td>
</tr>
<tr>
<td>target</td>
<td>INTEGER</td>
<td>The identification number of the target associated with the microbatch.</td>
</tr>
<tr>
<td>load_spec</td>
<td>INTEGER</td>
<td>The identification number of the load spec associated with the microbatch.</td>
</tr>
<tr>
<td>target_columns</td>
<td>VARCHAR</td>
<td>The table columns associated with the</td>
</tr>
</tbody>
</table>
### Column Names, Data Types, and Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rejection_schema</td>
<td>VARCHAR</td>
<td>The schema that contains the rejection table.</td>
</tr>
<tr>
<td>rejection_table</td>
<td>VARCHAR</td>
<td>The table where Vertica stores messages that are rejected by the database.</td>
</tr>
<tr>
<td>enabled</td>
<td>BOOLEAN</td>
<td>When TRUE, the microbatch is enabled for use.</td>
</tr>
</tbody>
</table>

### Examples

This example shows a row from a typical stream_microbatches table.

```
SELECT * FROM stream_config.stream_microbatches;
- [ RECORD 1 ]-----------
  id | 1
  mbatch | mbatch1
  target | 1
  load_spec | 750001
  target_columns | 
  rejection_schema | public
  rejection_table | rejected_messages
  enabled | t
```

### stream_scheduler

This table contains metadata related to a single scheduler.

This table was renamed from kafka_config.kafka_scheduler.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR</td>
<td>The version of the scheduler.</td>
</tr>
<tr>
<td>frame_duration</td>
<td>INTERVAL</td>
<td>The length of time of the frame. The default is 00:00:10.</td>
</tr>
<tr>
<td>resource_pool</td>
<td>VARCHAR</td>
<td>The resource pool associated with this scheduler.</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>config_refresh</td>
<td>INTERVAL</td>
<td>The interval of time that the scheduler runs before applying any changes to its metadata, such as, changes made using the -update option.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, refer to -config-refresh in Scheduler Tool Options.</td>
</tr>
<tr>
<td>new_source_policy</td>
<td>VARCHAR</td>
<td>When during the frame that the source runs. Set this value with the -new-source-policy in Source Tool Options.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid Values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FAIR: Takes the average length of time from the previous batches and schedules itself appropriately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• START: Runs all new sources at the beginning of the frame. In this case, Vertica gives the minimal amount of time to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• END: Runs all new sources starting at the end of the frame. In this case, Vertica gives the maximum amount of time to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: FAIR</td>
</tr>
<tr>
<td>eof_timeout_ms</td>
<td>INTEGER</td>
<td>The maximum amount of time the scheduler waits for data from the source before ending the batch.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 1000</td>
</tr>
<tr>
<td>pushback_policy</td>
<td>VARCHAR</td>
<td>(Not currently used.) How Vertica handles delays for microbatches that continually fail.</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid Values:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FLAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- LINEAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXPONENTIAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default Value:</strong> LINEAR</td>
</tr>
<tr>
<td>pushback_max_count</td>
<td>INTEGER</td>
<td>(Currently not used.) The maximum number of times a microbatch can fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>before Vertica terminates it.</td>
</tr>
<tr>
<td>auto_sync</td>
<td>BOOLEAN</td>
<td>When TRUE, the scheduler automatically synchronizes source information with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>host clusters. For more information, refer to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Automatically Copying Data From Kafka.][3]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default Value:</strong> TRUE</td>
</tr>
</tbody>
</table>

### Examples

This example shows a typical row in the stream_scheduler table.

```
SELECT * FROM stream_config.stream_scheduler;
-|[ RECORD 1 ]---------------------------------------
    version | v8.0.0
    frame_duration | 00:00:10
    resource_pool | kafka_default_pool
    config_refresh | 00:05
    new_source_policy | FAIR
    eof_timeout_ms | 10000
    pushback_policy | LINEAR
    pushback_max_count | 5
    auto_sync | t
```
**stream_scheduler_history**

This table shows the history of launched scheduler instances.

This table was renamed from kafka_config.kafka_scheduler_history.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elected_leader_time</td>
<td>TIMESTAMP</td>
<td>The time when this instance took began scheduling operations.</td>
</tr>
<tr>
<td>host</td>
<td>VARCHAR</td>
<td>The host name of the machine running the scheduler instance.</td>
</tr>
<tr>
<td>launcher</td>
<td>VARCHAR</td>
<td>The name of the currently active scheduler instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Default Value:</strong> NULL</td>
</tr>
<tr>
<td>scheduler_id</td>
<td>INTEGER</td>
<td>The identification number of the scheduler.</td>
</tr>
<tr>
<td>version</td>
<td>VARCHAR</td>
<td>The version of the scheduler.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows typical rows from the stream_scheduler_history table.

```sql
SELECT * FROM stream_config.stream_scheduler_history;
```

<table>
<thead>
<tr>
<th>elected_leader_time</th>
<th>host</th>
<th>launcher</th>
<th>scheduler_id</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-26 13:19:42.692</td>
<td>10.20.100.62</td>
<td></td>
<td>0</td>
<td>v8.0.0</td>
</tr>
<tr>
<td>2016-07-26 13:54:37.715</td>
<td>10.20.100.62</td>
<td></td>
<td>1</td>
<td>v8.0.0</td>
</tr>
<tr>
<td>2016-07-26 13:56:06.785</td>
<td>10.20.100.62</td>
<td></td>
<td>2</td>
<td>v8.0.0</td>
</tr>
<tr>
<td>2016-07-26 13:56:56.033</td>
<td>10.20.100.62</td>
<td>SchedulerInstance</td>
<td>3</td>
<td>v8.0.0</td>
</tr>
<tr>
<td>2016-07-26 15:51:20.513</td>
<td>10.20.100.62</td>
<td>SchedulerInstance</td>
<td>4</td>
<td>v8.0.0</td>
</tr>
<tr>
<td>2016-07-26 15:51:35.111</td>
<td>10.20.100.62</td>
<td>SchedulerInstance</td>
<td>5</td>
<td>v8.0.0</td>
</tr>
</tbody>
</table>

(6 rows)

**stream_sources**

This table contains metadata related to data streaming sources.
This table was formerly named `kafka_config.kafka_scheduler`.

### Column Data Type Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>The identification number of the source</td>
</tr>
<tr>
<td>source</td>
<td>VARCHAR</td>
<td>The name of the source.</td>
</tr>
<tr>
<td>cluster</td>
<td>INTEGER</td>
<td>The identification number of the cluster associated with the source.</td>
</tr>
<tr>
<td>partitions</td>
<td>INTEGER</td>
<td>The number of partitions in the source.</td>
</tr>
<tr>
<td>enabled</td>
<td>BOOLEAN</td>
<td>When TRUE, the source is enabled for use.</td>
</tr>
</tbody>
</table>

#### Examples

This example shows a typical row from the `stream_sources` table.

```sql
select * from stream_config.stream_sources;
```

- [ RECORD 1 ]---------
  id | 1
  source | SourceFeed1
  cluster | 1
  partitions | 1
  enabled | t

- [ RECORD 2 ]---------
  id | 250001
  source | SourceFeed2
  cluster | 1
  partitions | 1
  enabled | t

---

**stream_targets**

This table contains the metadata for all Vertica target tables.

The table was formerly named `kafka_config.kafka_targets`.

### Column Data Type Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>The identification number of the target table</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>target_schema</td>
<td>VARCHAR</td>
<td>The name of the schema for the target table.</td>
</tr>
<tr>
<td>target_table</td>
<td>VARCHAR</td>
<td>The name of the target table.</td>
</tr>
</tbody>
</table>

**Examples**

This example shows typical rows from the stream_tables table.

```sql
=> SELECT * FROM stream_config.stream_targets;
- [ RECORD 1 ]-+---------------------+-
id   | 1
| target_schema | public
| target_table  | stream_flex1
- [ RECORD 2 ]-+---------------------+-
id   | 2
| target_schema | public
| target_table  | stream_flex2
```
Integrating with Apache Spark

Welcome to the Vertica Vertica Connector for Apache Spark Guide.

The Vertica Connector for Apache Spark is a fast parallel connector that transfers data between the Vertica Analytics Platform and Apache Spark. This feature lets you use Spark to pre-process data for Vertica and to use Vertica data in your Spark application.

Apache Spark is an open-source, general-purpose cluster-computing framework. It evolved as a faster, multi-stage, in-memory alternative to the two stage, disk-based Map Reduce framework offered by Hadoop. The Spark framework is based on Resilient Distributed Datasets (RDDs), which are logical collections of data partitioned across machines. Spark is typically used in upstream workloads to process data before loading it in Vertica for interactive analytics. It can also be used downstream of Vertica, where data pre-processed by Vertica is then moved into Spark for further transformation.

Using the Vertica Vertica Connector for Apache Spark, you can:

- Move large volumes of data from Spark DataFrames to Vertica tables; the connector allows you to write Spark DataFrames to Vertica tables.

- Move data from Vertica to Spark RDDs or DataFrames for use with Python, R, Scala and Java. The connector efficiently pushes down column selection and predicate filtering to Vertica before loading the data.
Audience

This book is intended for anyone who wants to transfer data between a Vertica database and an Apache Spark cluster.
Prerequisites and Compatibility

This document assumes that you have installed and configured Vertica as described in Installing Vertica and the Configuring the Database section of the Administrator's Guide. You must also have installed your Apache Spark clusters.

To save data from Spark to Vertica, you must have an HDFS cluster for an intermediate staging location. Your Vertica database must be configured to read data from this HDFS cluster. See Reading Directly from HDFS in Integrating with Apache Hadoop for more information.

For details on installing and using Apache Spark and Apache Hadoop, see the Apache Spark web site, the Apache Hadoop website, or your Hadoop vendor's installation documentation.

For supported versions of Apache Spark and Apache Hadoop see the following sections in the Supported Platforms guide:

- Vertica Integration for Apache Spark
- Vertica Integrations for Hadoop
Getting the Spark Connector

The Vertica Connector for Apache Spark is packaged as a JAR file. In addition to the connector JAR file, you also need the Vertica JDBC client library. The connector uses this library to connect to the Vertica database.

You can download the Spark Connector from the myVertica portal in the Downloads section. The Vertica JDBC client library is also available from the myVertica portal in the client driver downloads section.

Choosing the Correct Connector Version

The connector JAR file is compatible with specific versions of Vertica, Apache Spark, and Scala. The connector is compatible with versions of Vertica later than 8.0. See Vertica Integration for Apache Spark in the Supported Platforms document.

You must download the connector file that matches the versions in your installation. For example, suppose you want to connect your Vertica 8.1.0 database to your Spark 2.0 cluster which has Scala version 2.11 installed on it. Then you need the connector JAR file named vertica-8.1.0_spark2.0_scala2.11.jar.

Note: The Vertica version number in the connector's file name refers to the Vertica server version in which the connector was released. Connector versions may be backwards compatible with earlier versions of the Vertica server. Vertica Integration for Apache Spark in Supported Platforms tells you which versions of Vertica the latest connector supports.

You can determine your Spark and Scala version by starting a Spark shell:

```
$ spark-shell
SPARK_MAJOR_VERSION is set to 2, using Spark2
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://node01:4040
Spark context available as 'sc' (master = local[*], app id = local-1488824765565).
Spark session available as 'spark'.
Welcome to
/
|\ _ _ _ _ _ _/_/
|\ \ \ \ \ \ \ \ /
|\ __\ __\ __\ __\ __\ __\ __\ __\ __\ version 2.0.0.2.5.3.0-37
|\ /
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
The startup messages contain the version numbers of both Spark and Scala (shown in bold in the previous example for clarity).
Deploying the Vertica Connector for Apache Spark

Once you have downloaded the connector and JDBC library JAR files, you can deploy them to your Spark cluster in two ways:

- Include the connector and Vertica JDBC JAR files using the "--jars" option when invoking spark-submit or spark-shell.

- Deploy the connector to a Spark cluster so that all Spark applications have access across all nodes.

Copying the Connector for Use with Spark Submit or Spark Shell

1. Log on as the Spark user on any Spark machine.

2. Copy both the Vertica Spark Connector and Vertica JDBC Driver JAR files from the package to your local Spark directory.

3. Then, run the connector in any of the following ways (replace the version number in the jar names with your version number):

   - Run a Spark Application using spark-submit
     (http://spark.apache.org/docs/latest/submitting-applications.html#launching-applications-with-spark-submit). You must run the command on the node with the Vertica JAR files.

     ```
     spark-submit --jars vertica-8.1.0_spark2.0 Scala2.11.jar,vertica-jdbc-8.1.0-0.jar other options SparkApplication.jar
     ```

     **Note:** Do not include a space before or after the comma in the --jars argument.

   - Use the Interactive spark-shell (http://spark.apache.org/docs/latest/programming-guide.html#using-the-shell). You must run the command on the node with the Vertica JAR files:
Deploying the Connector to a Spark Cluster

You can optionally deploy the JAR files to a Spark cluster. This approach gives all applications (such as shell and submit) access; it does not require specifying them on the command line.

To deploy to the Spark cluster:

1. Copy the files to a common path on all Spark machines.

2. Add the path for the connector and JDBC driver to your conf/spark-defaults.conf, and restart the Spark Master. For example, modify the spark.jars line by adding the connector and JDBC JARS as follows (replace paths and version numbers with your values):

```
spark.jars /JAR_file_Path/vertica-8.1.0_spark2.0_scala2.11.jar,JAR_file_Path/vertica-jdbc-8.1.0.jar
```
Saving an Apache Spark DataFrame to a Vertica Table

The Vertica Connector for Apache Spark copies data partitions distributed across multiple Spark worker-nodes into a temporary location in HDFS. Vertica then reads the data from HDFS. These data transfers use parallel reads and writes, letting the connector efficiently load large volumes of data from Spark to Vertica.

Save Options

When writing Spark DataFrames to Vertica, you specify a Vertica target table. You also set how the data is saved using the DataFrame SaveMode. The valid values for the SaveMode are:

- **SaveMode.Overwrite** overwrites or create new table.

  Note: The existing table is dropped whether the save succeeds or not, except when the connector is asked to load a DataFrame that contains zero rows.

- **SaveMode.Append** appends data to an existing table or creates the table if it does not exist.

- **SaveMode.ErrorIfExists** creates a new table if the table does not exist, otherwise returns an error.

- **SaveMode.Ignore** creates a new table if the table does not exist, otherwise it does not save the data and does not return an error.

The `save()` operation never results in a partial or duplicate save. The connector either saves the DataFrame in its entirety to the target table successfully or it aborts the save. In case of failures, to avoid leaving the target table in an inconsistent state, the connector:

1. Saves the data to HDFS as an intermediate staging location
2. Safely copies it into the actual target table in Vertica
Rejected Rows

For bulk loads, the connector API provides user control to specify a tolerance for rejected rows. You can specify the user tolerance as a parameter (see Parameters in Saving Spark Data to Vertica Using the DefaultSource API). If the number of rejected rows falls within the tolerance level, the save completes successfully. Otherwise, the connector aborts the save and reports an error.

Job Status

The connector creates a Vertica table, S2V_JOB_STATUS_USER_$USERNAME$, to report the status of each job. When the save to Vertica starts, the connector writes the unique job_id to the Spark log. After the Spark job is started, users can consult this table for the job status based on the job_id. The S2V_JOB_STATUS_USER_$USERNAME$ has the following columns:

- target_table_schema
- target_table_name
- save_mode
- job_name
- start_time
- all_done
- success
- percent_failed_rows

The table indicates the start time, unique job name, date, percentage of rows that failed, and final outcome of the save job (success=T/F). The column all_done indicates the Spark job finished without errors and the column success indicates the data was saved to the Vertica table (rejected rows must fall within specified tolerance).
Saving Spark Data to Vertica Using the DefaultSource API

The Vertica Connector for Apache Spark provides the com.vertica.spark.datasource.DefaultSource API to simplify writing data from a Spark DataFrame to a Vertica table using the Spark df.write.format() method. The DefaultSource API provides generic key-value options for configuring the database connection and tuning parameters as well as other options.

Requirements

The following requirements apply to using the connector to save Spark data to Vertica.

- When you append to an existing Vertica table (using SaveMode.Append), your DataFrame must have the same column types, column order, and number of columns as the Vertica table.

- The specified Vertica username must have CREATE privileges on the Vertica schema. The connector defaults to "public" schema, but any schema name can be provided as an option by using the "dbschema" optional argument.

- A Spark SaveMode must to be specified (Append, Overwrite, Ignore, ErrorIfExists). See Saving an Apache Spark DataFrame to a Vertica Table and https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/sql/SaveMode.html.

Limitations

The following limitations apply to using the connector to save Spark data to Vertica.

- Supported Data Types—The Spark DataFrame cannot have complex types such as Maps, Structs, and Arrays. Vertica currently supports all other Spark data types. The connector cannot save a DataFrame that contains any of these types.

- Target Table Limitations—The Vertica target table specified cannot be a view or a temporary table.
Null Values in Spark and Vertica—Spark’s long type is converted to Vertica’s INTEGER type when data moves from Spark to Vertica. Vertica uses the value \(-2^{63}\) (-29223372036854775808) to represent a NULL value. Spark does not treat this value in any special way. If you save \(-2^{63}\) from Spark to Vertica then it is stored as NULL in Vertica.

SQL Strings—Length of Strings. Currently the connector converts all Spark SQL Strings to VARCHAR(strlen) for Vertica. The ‘strlen’ parameter is a user option that defaults to 1024. If strlen is set to a value greater than 65000, Spark SQL Strings are instead converted to LONG VARCHAR(strlen). Values in the range 1 to 32,000,000 are valid.

When using SaveMode.Append, the existing Vertica table should have the corresponding column types for Spark SQL Strings declared as either VARCHAR or LONG VARCHAR.

Spark SQL Strings longer than strlen are always truncated when saving to Vertica.

Float Values—Float Values Represented Differently. Because float values are approximations, the value of a float can be different when it is moved from Spark to Vertica. If you require more precision, then use a more precise data type in Spark, such as double or decimal.

Gregorian/Julian Calendar Issues—Inconsistent dates before the year 1583. The ORC file writer used to stage the data to in HDFS converts dates before 1583 from the Gregorian calendar to the Julian calendar. However Vertica does not perform this conversion. If your data in Spark contains dates before 1583, then the values in Spark and the corresponding values in Vertica can differ by up to 10 days. This difference applies to both DATE and TIMESTAMP values.

Wrong results for Decimal Type. If your version of Vertica is prior to 7.2 SP3, then decimal numbers are incorrectly loaded from Spark to Vertica. Upgrade to a newer version of Vertica to resolve this issue.

Different Jobs Writing to the Same Table in Overwrite Mode cause error:—ERROR: java.sql.SQLNonTransientException: [Vertica][VJDBC](3007) ERROR: DDL statement interfered with this statement. If you have different jobs writing to the same table and are using Overwrite mode then errors can occur. For example, Job 1 starts and begins writing to the Vertica table. Then Job 2 begins and is set to write to the same table. Because the mode is Overwrite, it drops the table to which Job 1 is writing, causing an error. Do not use different jobs to write to the same table when using overwrite mode.

Files not cleaned up if process interrupted.—The files written on by Spark to HDFS are normally cleaned up when the job completes. However, if something interrupts the process
(such as a network failure), then files may be left behind and you should manually cleanup the HDFS directory. The HDFS directory is the one provided in the hdfs_url path, and the specific subdirectory is $S2V_{jobxyz}$ as reported by the connector.

- **Error if the DataFrame has zero rows**—The connector returns an error message asking you to check whether the DataFrame is empty. This error occurs before the connector begins loading data into Vertica. The data in the targeted Vertica table is not altered, even if mode was set to SaveMode.Overwrite (which would normally delete the table).

## Parameters

To save your Spark DataFrame to Vertica, you must specify the following required parameters as options to `sqlContext.createDataFrame(...).format("com.vertica.spark.datasource.DefaultSource").options(...)`:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The name of the target Vertica table to save your Spark DataFrame.</td>
</tr>
<tr>
<td>db</td>
<td>The name of the Vertica Database</td>
</tr>
<tr>
<td>user</td>
<td>The name of the Vertica user. This user must have CREATE and INSERT privileges in the Vertica schema. The schema defaults to “public”, but may be changed using the “dbschema” optional tuning parameter.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the Vertica user.</td>
</tr>
<tr>
<td>host</td>
<td>The hostname of a Vertica node. This value is used to make the initial connection to Vertica and look up all the other Vertica node IP addresses. You can provide a specific IP address or a resolvable name such as myhostname.com.</td>
</tr>
<tr>
<td>hdfs_url</td>
<td>The fully-qualified path to a directory in HDFS that will be used as a data staging area. For example, <code>hdfs://myhost:8020/data/test</code>. The connector first saves the DataFrame in its entirety to this location before loading into Vertica. The data is saved in ORC format, and the files are saved in a directory specific to each job. This directory is then is deleted when the job completes. Note that you need additional configuration changes for to use HDFS. You must first configure all nodes to use HDFS. See <a href="#">Configuring the hdfs Scheme</a>.</td>
</tr>
</tbody>
</table>
In addition to the required parameters, you can optionally specify the following parameters as options to `sqlContext.createDataFrame(...).format("com.vertica.spark.datasource.DefaultSource").options(...)`. | Parameter     | Description                                                                 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dbschema</td>
<td>The schema space for the Vertica table. Default Value: public</td>
</tr>
<tr>
<td>port</td>
<td>The Vertica Port. Default value: 5433</td>
</tr>
<tr>
<td>failed_rows_percent_tolerance</td>
<td>The tolerance level for failed rows, as a percentage. For example, to specify that the job fail if greater than 10% of the rows are rejected, specify this value as 0.10 for 10% tolerance. Default Value: 0.00</td>
</tr>
<tr>
<td>strlen</td>
<td>The string length. Use this option to increase (or decrease) the default length when saving Spark StringType to Vertica VARCHAR type. Default Value: 1024</td>
</tr>
<tr>
<td>web_hdfs_url</td>
<td>The fully-qualified path to a directory in HDFS that will be used by Vertica to retrieve the data. For example, <code>webhdfs://myserver:50070/data/test</code>. You must use this option (in addition to “hdfs_url”) if the Vertica nodes are not configured for HDFS access. See <a href="#">HDFS and WebHDFS</a> below.</td>
</tr>
<tr>
<td>fileformat</td>
<td>The format for the intermediate data file written to HDFS. Supported values are:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- &quot;orc&quot; (default)</td>
</tr>
<tr>
<td></td>
<td>- &quot;parquet&quot;</td>
</tr>
</tbody>
</table>

See [Intermediate Data File Format Settings](#) below for more information.
HDFS and WebHDFS

The connector transfers data from Spark to an HDFS directory before moving it to Vertica. Vertica can access data in the HDFS directory either directly using the hdfs scheme, or through the web-based webhdfs scheme. For greater performance and reliability, Vertica should use direct HDFS access whenever possible.

You must configure your Vertica cluster before it can directly access data stored in HDFS. See Configuring the hdfs Scheme in the Integrating with Apache Hadoop.

By default, the connector uses the hdfs protocol when possible. It falls back to the webhdfs protocol in several cases:

- If you do not have your nodes configured for direct HDFS access.
- If the data stored in the HDFS directory is encrypted. The library that Vertica uses to read directly from HDFS cannot decrypt this data. The webhdfs scheme transparently decrypts the data for Vertica.

For Vertica to fall back to webhdfs, you must verify that webhdfs.enabled set to true on your HDFS cluster.

Intermediate Data File Format Settings

When Spark sends data to Vertica, it writes the data to intermediate files stored in HDFS. Vertica then reads these files off of HDFS. By default, Spark writes these data files in ORC format. You can choose to have it use Parquet format instead by setting the fileformat parameter to "parquet".

You usually do not directly access these files, so you normally do not care about their format. The Spark connector automatically deletes these files after Vertica is through with them.

Important: Testing shows that Spark writes ORC files faster than Parquet files. Writing of the data files makes up a large portion of the time it takes to transfer data from Spark to Vertica. Using the format that takes Spark the least amount of time to write results in faster transfers. For this reason, Vertica recommends you use the default ORC format for the intermediate files unless you are required to use Parquet format files.
Quick Start with Code Example

You can use the spark-shell and some brief Scala code to verify that the connector can write data from Spark to Vertica.

1. Start the spark-shell and include both the Vertica JDBC Driver and the Vertica Spark Connector JAR files in the jars argument. Then, specify any other Spark options you normally use:

   ```bash
   spark-shell --jars vertica-8.1.0_spark2.0_scala2.11.jar,vertica-jdbc-8.1.0-0.jar other options
   ```

2. The following code creates a DataFrame in Spark and saves it to a new Vertica table using the connector:

   a. Modify the host, db, user, password, and table settings in the following example to match your Vertica instance.

   b. Paste the following code into your Spark shell.

   ```scala
   // S2V_basic.scala
   import org.apache.spark.sql.types._
   import org.apache.spark.sql.{DataFrame, Row, SQLContext, SaveMode}
   import com.vertica.spark._

   // Create a sample DataFrame and save it to Vertica
   val rows = sc.parallelize(Array(
     Row(1,"hello", true),
     Row(2,"goodbye", false)
   ))
   val schema = StructType(Array(
     StructField("id",IntegerType, false),
     StructField("message",StringType,true),
     StructField("still_here",BooleanType,true)
   ))

   // Note: Spark's API changed between version 1.6 and 2.0. In version 1.6
   // you use sqlContext to create a DataFrame. In 2.0, you use the spark object.
   // val df = sqlContext.createDataFrame(rows, schema) // Spark 1.6
   val df = spark.createDataFrame(rows, schema) // Spark 2.0

   // View the sample data and schema
   df.show
   df.schema

   // Setup the user options, defaults are shown where applicable for optional values.
   // Replace the values in italics with the settings for your Vertica instance.
   val opts: Map[String, String] = Map(
     "table" -> "VerticaTableName",
   )
   ```
"db" -> "VerticaDatabaseName",
"user" -> "VerticaDatabaseUser",
"password" -> "VerticaDatabasePassword",
"host" -> "VerticaHostName",
"hdfs_url" -> "hdfs://HDFSNameNode:9000/user/hduser/someDirectory",
"web_hdfs_url" -> "webhdfs://HDFSNameNode:50070/user/hduser/someDirectory"
// "failed_rows_percent_tolerance"-> "0.00"  // OPTIONAL (default val shown)
// "dbschema" -> "public"  // OPTIONAL (default val shown)
// "port" -> "5433"  // OPTIONAL (default val shown)
// "strlen" -> "1024"  // OPTIONAL (default val shown)
// "fileformat" -> "orc"  // OPTIONAL (default val shown)

// SaveMode can be either Overwrite, Append, ErrorIfExists, Ignore
val mode = SaveMode.Overwrite

// save the DataFrame via Spark's Datasource API
df.write.format("com.vertica.spark.datasource.DefaultSource").options(opts).mode(mode).save()
Loading Vertica Data into a Spark DataFrame or RDD

The Vertica Connector for Apache Spark includes APIs to simplify loading Vertica table data efficiently with an optimized parallel data-reader:

- `com.vertica.spark.datasource.DefaultSource` — The data source API, which is used for writing to Vertica and is also optimized for loading data into a DataFrame.
- `RDD API com.vertica.spark.rdd.VerticaRDD` — This API simplifies creating an RDD object based on a Vertica table or view.

Typically, Vertica tables are segmented across multiple nodes. Using the Spark connector, you invoke a parallel data reader to efficiently read data from Vertica by minimizing data movement between Vertica nodes. The DataFrame reader supports pushing column and row filtering to Vertica to avoid transferring large volumes of Vertica data into the Spark in-memory data structures.

**Important:** For the best possible performance, segment your Vertica table by hash on one or more attributes that return integer values. Micro Focus does not recommend segmenting by a custom expression, because doing so can result in lower performance than segmenting by hash. See Hash Segmentation Clause for more details. An example of creating a table segmented by hash:

```sql
create table example(a integer, b integer) segmented by hash(a) all nodes;
```

Loading Vertica Table Segments into the Spark DataFrames and RDD Partitions

All Spark RDDs and DataFrames require you to define the number of partitions. You can define an arbitrary number of partitions. The Vertica Spark connector library automatically generates the hash-intervals for each partition and intervals. When you segment the table by a proper hash expression on one or more of its columns, these intervals minimize cross-node data shuffling (w) inside Vertica and data skew.

No cross-node data shuffling occurs inside Vertica when both of the following conditions exist:
- The Vertica cluster has $N$ nodes.

- The number of the Spark partitions is $P \times N$ ($P$ is an integer greater than 0).

Therefore, you can use the full network throughput for data transfer, achieving the best performance. The following figure shows 8 Spark partitions defined over 4 Vertica segmentations.

The next figure shows another example where the number of Spark partitions is smaller (two) than the number of Vertica segmentations (four).

According to Vertica benchmark tests, a setting of $P=4$ achieves the best loading performance. For example, when Vertica has 4 nodes, the number of Spark partitions recommended is 16 ($P = 4$, Nodes = 4, $P \times N = 16$ partitions).

During Spark job execution, each Spark partition executes inside a task. Each task launches a JDBC connection to a Vertica node. This connection contacts the Vertica node that stores the segment containing the data of the Spark partition. The Spark task then issues a query to the Vertica node and fetches the query result into a VerticaRDD/DataFrame.

For an unsegmented Vertica table, the connector replicates the table onto multiple Vertica nodes. The Spark partition is defined as a range on the sorted columns of the table. Spark tasks send the queries to different replicas for load-balancing. The next figure shows Spark partitions defined over an unsegmented table.

The following figure illustrates the runtime behavior of the Vertica to Spark connector. Spark tasks containing VerticaRDD/DataFrame partitions fetch data from Vertica through JDBC connections. Those queries that are used to fetch data are based on the computed ranges of hash values.
Spark Numeric Value Limitation

Both Vertica and Spark support variable-precision NUMERIC values. Vertica's `NUMERIC` values support more digits of precision (up to 1024) than Spark does (38 digits). If you attempt to copy a Vertica table to Spark that has a NUMERIC column with more than 38 digits of precision, the `VerticaDataSourceRDD` class throws an error similar to the following:

```java
java.lang.IllegalArgumentException: requirement failed: Decimal precision 41 exceeds max precision 38
at scala.Predef$.require(Predef.scala:224)
at org.apache.spark.sql.types.Decimal.set(Decimal.scala:113)
at org.apache.spark.sql.types.Decimal$.apply(Decimal.scala:426)
at com.vertica.spark.datasource.VerticaDataSourceRDD$$anon$1.getNext(VerticaRDD.scala:382)
```

Loading Vertica Data into a Spark DataFrame
Using the Vertica Data Source API

The Vertica Connector for Apache Spark data source API supports both parallel write and read operations. The following code sample illustrates how you can create an in-memory DataFrame by invoking `SQLContext.read` function, using Vertica's `com.vertica.spark.datasource.DefaultSource` formatter.

Parameters

To connect to Vertica, you must specify the following parameters for the options in
`sqlContext.read.format
("com.vertica.spark.datasource.DefaultSource").options(...)`. 
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The name of the target Vertica table or view to save your Spark DataFrame. Note that the Vertica table must be segmented by hash or by an expression that returns non-negative integer values. Also, if the Vertica table is an external table, the underlying file(s) on which the external table is based must be accessible on all Vertica nodes.</td>
</tr>
<tr>
<td>db</td>
<td>The name of the Vertica Database</td>
</tr>
<tr>
<td>user</td>
<td>The name of the Vertica user. This user must have CREATE and INSERT privileges in the Vertica schema. The schema defaults to “public”, but may be changed using the dbschema optional parameter.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the Vertica user.</td>
</tr>
<tr>
<td>host</td>
<td>The hostname of a Vertica node. This value is used to make the initial connection to Vertica and look up all the other Vertica node IPs. You can provide a specific IP address or resolvable name such as myhostname.com.</td>
</tr>
<tr>
<td>numPartitions</td>
<td>Optional. The number of Spark partitions for the load from the Vertica job, each partition creates a JDBC connection. Default Value: 16.</td>
</tr>
<tr>
<td>dbschema</td>
<td>Optional. The schema space for the Vertica table. Default value: public.</td>
</tr>
<tr>
<td>port</td>
<td>Optional. The Vertica Port. Default value: 5433</td>
</tr>
</tbody>
</table>

**Example: Load Data into a DataFrame**

The following example demonstrates reading the content of a Vertica table named test into a DataFrame.

```scala
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.sql._
import org.apache.spark.SparkConf

// Note: the following is deprecated in Spark 2.0. It will warn you to use // SparkSession instead.
val sqlContext = new SQLContext(sc)

val table = "test"
val db = "myDB"
```
val user = "myUser"
val password = "myPassword"
val host = "myVerticaHost"
val part = "12";

val opt = Map("host" -> host, "table" -> table, "db" -> db, "numPartitions" -> part, "user" -> user, "password" -> password)

val df = sqlContext.read.format("com.vertica.spark.datasource.DefaultSource").options(opt).load()

## Column Selection and Filter Push Down

Column selections and row filters applied on the DataFrame are pushed down into Vertica. Refer to the Spark SQL API for more details about applying column selections and row filters. The following example illustrates how you can load a column containing filtered rows as a DataFrame:

```
val c = df.select("a").filter("a > 5").count
```

**Note:** When filtering it is important to specify the correct type. For example, single-quoted values are treated as strings, so if you wrote the above filter code as `filter("a > '5"`)` then the filter is not pushed down, because the target column is an integer and the value '5' when single-quoted is treated as a string.

Additionally, when using other types such as dates, cast the value as a date type, such as:

```
filter("c1 >= cast('2010-1-2' as date) rather than filter("c1 >= '2010-1-2'"). The latter example does not push down.
```

Additional examples of valid filters:

- `df.filter("id<=3").groupBy("id").sum("id").show`
- `df.filter($"c".like("str%")).show // has to be varchar type to be pushed down`
- `df.filter($"c".rlike("str")).show // has to be varchar type to be pushed down`
- `df.filter($"id".isin(3,5)).show`
Considerations When Using the Spark DataFrame Filter Method with Vertica

Be aware of the following considerations when using the DataFrame's filter method when loading data from Vertica:

- Vertica supports the case insensitive terms “inf” and “infinity” to refer to INFINITY in SQL queries. However, when filtering using Spark’s filter method you must instead use the term (case sensitive) “Infinity”.

- When filtering on Boolean values, do not use "is true/false" or "is not true/false" instead, cast the value as a Boolean, or use the equals operator, for example:
  - `df.filter("c = 1").show`
  - `df.filter("c = True").show`
  - `df.filter("c = False").show`

Example: Creating a DataFrame Using the Vertica Data Source

Use the following example to learn how to create a Spark DataFrame from a Vertica table:

1. Create a sample table in Vertica:

   ```sql
   => CREATE TABLE test (a int, b int, c int, d varchar);
   => INSERT INTO test VALUES (1, 3, 5, 'odds');
   => INSERT INTO test VALUES (10, 14, 8, 'evens');
   => INSERT INTO test VALUES (11, 13, 19, 'odds');
   => COMMIT;
   ```

2. Modify the database connection details (host, db, table, user, and password) in the code below with your Vertica connection details.

   ```java
   import org.apache.spark.sql.SQLContext
   import org.apache.spark.SparkConf
   import org.apache.spark.SparkContext

   val conf = new SparkConf().setAppName("vertica-spark-connector-testing").setMaster("local[1]")
   val sc = new SparkContext(conf)
   ```
3. Run the example code in a Spark shell. Start the shell, then paste in your modified code.

**Loading Vertica Data into a Spark RDD Using the Vertica RDD API**

Use the Vertica RDD API to load Vertica table data into a Spark RDD. You create a VerticaRDD object either by initiating it or by calling one of the multiple RDD.create methods available. The following example uses VerticaRDD.create to create an RDD from Vertica data. In this example, the parameters of create() are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>A Vertica node host name</td>
</tr>
<tr>
<td>port</td>
<td>Vertica port number. The default value is 5433.</td>
</tr>
<tr>
<td>prop</td>
<td>A property object that include user name and password</td>
</tr>
<tr>
<td>table</td>
<td>Vertica table name, including the schema name, such as, “schema_name.table_name”.</td>
</tr>
<tr>
<td>dbname</td>
<td>Vertica database name</td>
</tr>
<tr>
<td>col</td>
<td>The column names of the Vertica table that will be loaded into Spark. An empty col array means all columns.</td>
</tr>
</tbody>
</table>
## Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numPartitions</td>
<td>The number of Spark partitions for the resulting VerticaRDD</td>
</tr>
<tr>
<td>mapRow</td>
<td>A function used to convert one row of JDBC results into the element data type T of VerticaRDD. The generated VerticaRDD has the type VerticaRDD[T]. For example:</td>
</tr>
<tr>
<td></td>
<td>val extractValues = (r: ResultSet) =&gt; { (r.getInt(1), r.getInt(2))</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>This function converts a tuple that contains two integers—for example, (101, 102), into an array of objects. Refer to the Spark JDBC RDD for more details about this parameter.</td>
</tr>
<tr>
<td>Note: This function must be serializeable by Spark.</td>
<td></td>
</tr>
</tbody>
</table>

### RDD Create Methods

The com.vertica.spark.rdd.VerticaRDD API has three different create methods that you can use to create a VerticaRDD object:

- `create(sc, connect, table, columns, numPartitions, mapRow)`
- `create(sc, connect, table, columns, numPartition, ipMap, mapRow)`
- `create(sc, host, port: Int = 5433, db, properties, table, columns, numPartitions, mapRow)`

The parameters for the Create methods:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>Spark Context object.</td>
</tr>
<tr>
<td>connect</td>
<td>A connection object that contains a function that returns an open JDBC Connection.</td>
</tr>
<tr>
<td>table</td>
<td>A string value for the database table name</td>
</tr>
<tr>
<td>columns</td>
<td>An Array[String] that contains the columns of the database table</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>that will be loaded. If an empty array, this RDD loads all columns.</td>
</tr>
<tr>
<td>numPartitions</td>
<td>An integer value specifying the number of partitions for the VerticaRDD.</td>
</tr>
<tr>
<td>ipMap</td>
<td>A Map [String, String] containing the optional map from private IP addresses to public IP addresses for all Vertica nodes. This map is used only when Vertica is installed on private IP addresses but is listening on both private and public IP addresses. You can also update each Vertica node's EXPORT_ADDRESS instead of providing the ipMap parameter if Vertica is running on both private and public IP addresses. When a database is installed, the export_address and node_address in the system NODES table are set to the same value. If you installed Vertica on a private address, you must set the export_address to a public address for each node. See Identify the Database or Nodes Used for Import/Export in the Administrator's Guide.</td>
</tr>
<tr>
<td>mapRow</td>
<td>A function from a ResultSet to a single row of the upi result types you want. This function should call only getInt, getString, etc; the RDD takes care of calling next. The default maps a ResultSet to an array of Object.</td>
</tr>
<tr>
<td>host</td>
<td>A string containing the host name (or IP address) of a Vertica node, that can be used to connect to Vertica.</td>
</tr>
<tr>
<td>port</td>
<td>An integer value specifying the number for Vertica connection. Default Value: 5433</td>
</tr>
<tr>
<td>properties</td>
<td>Properties object for JDBC connection properties, such as user, password.</td>
</tr>
</tbody>
</table>

**Example: Loading Vertica Table Data into a VerticaRDD**

This section contains complete example code used to create a VerticaRDD from a Vertica table, test whose definition is:

```java
create table test (a int, b int);
```
This program creates \texttt{VerticaRDD[(Int, Int)]} and calls \texttt{count} that returns the number of rows of table, test.

```scala
// V2S_rdd.scala
import com.vertica.spark.rdd.VerticaRDD
import java.util.Properties
import java.sql.ResultSet
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

val extractValues = (r: ResultSet) => {
  (r.getInt(1), r.getInt(2))
}

val conf = new SparkConf().setAppName("vertica-spark-connector-testing").setMaster("local[1]")
//val sc = new SparkContext(conf) // uncomment if not used in spark shell

val host = "VerticaHost"
val port = 5433
val db = "VerticaDB"
val prop = new Properties
prop.put("user", "VerticaUserName")
prop.put("password", "VerticaPassword")
val table = "test"
val cols = Array[String]()
val part = 12;
val data = VerticaRDD.create(sc, host, port, db, prop, table, cols, numPartitions = part, mapRow = extractValues)
val c = data.count
println("count:" + c)
```
Vertica Pulse

Welcome to the Vertica Pulse Guide. This book describes how to use Vertica Pulse.
Audience

This book is intended for anyone who wants to use the sentiment analysis features in Vertica.
**Prerequisites**

This document assumes that you have installed and configured Vertica as described in Installing Vertica and the Configuring the Database section of the Administrator's Guide.

Additionally, you must install a compatible Java JRE or JDK on all nodes in the cluster and configure Vertica to use Java for Java UDxs.

Please see the Vertica product documentation to learn more.
**Pulse Virtual Machine Quick Start**

These Quick Start instructions detail the minimal steps for installing and using Pulse with the Vertica Virtual Machine Image. Consult the complete documentation for detailed steps on installing Pulse on your own platform.

**Downloading and Installing Pulse**


2. Scroll down to the section "Download Vertica 7.1 Virtual Machines" and click the download link for your VM environment. These instructions assume you are installing the VMDK version - VMWare Server 2.0 and Workstation 7.0.

3. After the download completes, unzip the file.

4. Double-click the .vmx file in vmsrvr_64/Vertica 7.1.x x64 for VMware. The VM starts in your VMWare application.

5. You are automatically logged in as `dbadmin`. However, the password for the user (and root) is 'password'.

6. In the VM, select Applications > Accessories > Terminal to open a terminal.

7. In the terminal, type `admintools` to start the administration tools.

8. You are prompted for a license when admintools starts for the first time. To use the community edition license, simply click OK. You are then prompted to accept the EULA. Accept the EULA then exit admintools.

9. As `dbadmin`, using `vsql` on any node in the cluster, set the JavaBinaryForUDx Configuration Parameter (use `which java` to determine your java location):

   ```
   vsql -t -c "ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';"
   ```

10. Copy the Vertica Pulse install package to the VM then, as root, install the Pulse Package:

    ```
    rpm -Uvh /path/to/vertica-pulse.x86_64.xxx.rpm
    ```

**Note:** Only install Vertica Pulse on a single node. All Pulse functions are available on all nodes. However, the installation SQL scripts and user-dictionary loading script are only
available on the node on which you install the Pulse package.

11. As dbadmin, run the Pulse install script on the node on which you installed the Pulse Package:

```
vsq1 -f /opt/vertica/packages/pulse/ddl/install.sql
```

**Using Pulse**

1. Run a sentiment function:

```
select sentimentanalysis('Cookies are sweet.') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cookies</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)

Note: By default, VerticaPulse analyzes English text, however, you can also specify the language of the text being analyzed as an attribute of the sentimentanalysis() function. For example:

```
select sentimentanalysis('Cookies are sweet.', 'english') OVER(PARTITION BEST);
```

English and Spanish are the supported languages.
About the Vertica Pulse Package

Vertica Pulse provides a suite of functions that allow you to analyze and extract the sentiment from English and Spanish language text directly from your Vertica database.

Vertica Pulse features include:

- **Attribute based sentiment scoring** - Pulse scores the sentiment of attributes in a sentence. Attributes are generally nouns and are automatically discovered by Pulse. Pulse typically scores sentiment from a range of -1 (negative sentiment) to +1 (positive sentiment). A sentiment of 0 is considered neutral. Scoring individual attributes in a sentence instead of scoring the sentence as a whole provides a more granular analysis for the text. For example, consider the sentence "The quick brown fox jumped over the lazy dog." It would be difficult to score the sentiment on the sentence as a whole, but if you score on the attributes of fox and dog, you could say the sentiment on the fox was positive (the fox is quick), and the sentiment on the dog is negative (the dog is lazy).

- **Tuning to your domain** - Pulse provides functionality to recognize attributes that are specific to your domain. For example, you can add the name of your product or company to a 'white_list' so that it is discovered by Pulse.

- **Tuning of how sentiment is scored** - Pulse includes user-dictionaries of words that are used to help score sentiment. You can alter these user-dictionaries to fine tune the way your text is analyzed.

- **Filtering of attributes you are not interested in** - Pulse supports a special 'stop words' user-dictionary to indicate attributes that should not be analyzed. Alternately, you can choose to score sentiment only on attributes defined in your white_list.

- **Synonym mappings** - Pulse provides customizable mappings so that you can map synonyms to a base word, and then normalize the analysis for the synonyms to the base word. For example, you can map *Hewlett Packard* to *HP*.

Vertica Pulse requires that Java and the Vertica Java Support Package are installed on all nodes in the Vertica cluster.

Depending on the version of Pulse, it may support only one language (English or Spanish) or multiple languages (English and Spanish). For multilingual versions, Pulse can analyze each text row (for example a tweet) in the language of the text specified as argument, the language specified by the user as parameter or the default language. See **Multilingual Pulse** for details.
Installing or Upgrading Vertica Pulse

The Vertica Pulse Package requires that Java be installed prior to installing Vertica Pulse.

Vertica Pulse Package Version Requirements

Your server must be running version 7.1.x or later to run Pulse. Pulse must be installed on a Vertica node.

You can download the Vertica server package and from the Vertica Marketplace.

Installation Overview

1. Verify that your Vertica server version matches your Vertica Pulse version.

2. Install Java on all Hosts and set the JavaBinaryForUDx Vertica configuration parameter to your Java binary location. For example, using vsqI: ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java'

3. Install the Vertica Package on a single node in the cluster. The process is the same for installation or upgrade. You need only install it on a single node, but note that the SQL scripts used to install and uninstall the Pulse functions and the SQL script that creates pulse schema and the user-dictionaries tables are only available from the node on which you installed the Pulse package. The Pulse functions, once installed, are available on all nodes regardless if the package is installed on the node to which you are connecting.

4. Modify the jvm resource pool so that Pulse performs optimally on your system hardware.
Installing Java on Vertica Hosts

You must install a Java Virtual Machine (JVM) on every host in your Vertica cluster in order to run Pulse. Pulse requires a 64-bit Java Standard Edition 6, 7, or 8 (Java version 1.6, 1.7, 1.8) runtime. Both the Oracle JDK and openjdk are supported. You can choose to install either the Java Runtime Environment (JRE) or Java Development Kit (JDK), since the JDK also includes the JRE. See the Java Standard Edition (SE) Download Page to download an Oracle installation package for your Linux platform, or use your platforms packaging tool (such as yum or apt-get) to get a Java 1.6, 1.7, or 1.8 compatible version of open-jdk.

Once you have installed a JVM on each host, ensure that the java command is in the search path and calls the correct JVM by running the command:

```
java -version
```

This command should print something similar to:

```
java version "1.6.0_37"
Java(TM) SE Runtime Environment (build 1.6.0_37-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.12-b01, mixed mode)
```

Setting the JavaBinaryForUDx Configuration Parameter

The JavaBinaryForUDx configuration parameter tells Vertica where to look for the JRE to execute Java UDFs. After you have installed the JRE on all of the nodes in your cluster, you need to set this parameter to the absolute path of the Java executable. You can use the symbolic link that some Java installers create (for example /usr/bin/java). If the Java executable is in your shell search path, you can get the path of the Java executable by running the following command from the Linux command line shell:

```
$ which java
/usr/bin/java
```

If the java command is not in the shell search path, use the path to the Java executable in the directory where you installed the JRE. For example, if you installed the JRE in /usr/java/default (which is where the installation package supplied by Oracle installs the Java 1.6 JRE), the Java executable is /usr/java/default/bin/java.

You set the configuration parameter by executing the following statement as a database superuser:
ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';

See ALTER DATABASE for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the CONFIGURATION_PARAMETERS system table:

```sql
=> ALTER DATABASE mydb SET JavaBinaryForUDx = '/usr/bin/java';

See ALTER DATABASE for more information on setting configuration parameters.

To view the current setting of the configuration parameter, query the CONFIGURATION_PARAMETERS system table:

```sql
=> SELECT * FROM CONFIGURATION_PARAMETERS WHERE parameter_name = 'JavaBinaryForUDx';

<table>
<thead>
<tr>
<th>node_name</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter_name</td>
<td>JavaBinaryForUDx</td>
</tr>
<tr>
<td>current_value</td>
<td>'/usr/bin/java'</td>
</tr>
<tr>
<td>default_value</td>
<td></td>
</tr>
<tr>
<td>change_under_support_guidance</td>
<td>f</td>
</tr>
<tr>
<td>changeRequiresRestart</td>
<td>f</td>
</tr>
<tr>
<td>description</td>
<td>Path to the java binary for executing UDx written in Java</td>
</tr>
</tbody>
</table>
```

Once you have set the configuration parameter, Vertica will be able to find the Java executable on each node in your cluster in order to execute Java UDFs.

**Note:** Since the location of the Java executable is set by a single configuration parameter for the entire cluster, you must ensure that the path to the Java executable is the same across all of the nodes in the cluster.
Installing or Upgrading the Vertica Pulse Package on Your Host

After you install a JVM on all of the nodes in your cluster, you must install the Pulse Package on a single node. If upgrading, install the new package on the same host on which you previously installed the package. Pulse installation or upgrade is a two-step process:

1. Install/Update the RPM or DEB package for Pulse.
2. Run included sql scripts to install or update the Pulse functions and create the user dictionaries.

The Pulse install process installs the functions and schema required for sentiment analysis. You need only install it on a single node. However, be aware that the following SQL scripts are only available from the node on which you installed the Pulse package:

- SQL scripts used to install and uninstall the Pulse functions
- SQL script that populates and loads the dictionaries

You can access Pulse functions on all nodes, regardless if the package is installed on the node to which you are connecting.

Install or Upgrade the Pulse Package

When you upgrade or reinstall Pulse, it automatically uses port 5433 for vsq1. If you are using a different port, configure it using the command `export VSQL_PORT=<port_number>`.

1. Copy the RPM or DEB package to the node where you want to install or upgrade Pulse. If you are upgrading Pulse then copy the new package to the same node where you previously installed the Pulse package. The version of Vertica Pulse must match the version of the Vertica server. For example, if your Vertica server is version 7.1.0, then the VerticaPulse version must also be 7.1.0.

   If you are upgrading Pulse, you can find the currently-installed version number of Pulse with the command:
```
select lib_version, lib_sdk_version from user_libraries where lib_name = 'SentimentLib';
```

2. Log into the host and install the package.

   - For Red Hat, use:
     ```
sudo rpm -Uvh /path-to-package/vertica-pulse.x86_64.xxx.rpm
     ```
   - For Debian, use:
     ```
sudo dpkg -i /path-to-package/vertica-pulse.x86_64.xxx.deb
     ```

The Pulse Package is installed to /opt/vertica/packages/pulse.

After you install the package, you must run the appropriate SQL scripts to install or upgrade the Pulse functions and install the dictionary tables. Vertica automatically reloads any labeled user-defined dictionaries.

### Running the Pulse Install Script

Run the install script to install or upgrade the Pulse functions and schema for the dictionaries and mappings required for sentiment analysis. You must run the install script once on the node on which you installed the package. After you run the install script, then all nodes can use the Pulse functions.

**Important!** Before running the install script, you must set the JavaBinaryforUDx configuration parameter or the install script fails to install the Pulse functions. See [Installing Java on Vertica Hosts](#).

To run the install script:

1. As the dbadmin user, on the node on which you installed the Pulse RPM/DEB, run the `install.sh` script:
   ```
   bash /opt/vertica/packages/pulse/install.sh
   ```
   **Note:** You must run the install script for installs or upgrades.

2. The script installs/upgrades the Pulse functions:
3. If this is a fresh installation, then Modify the jvm Resource Pool to match your system hardware.
Tuning the jvm Resource Pool for Vertica Pulse

**Note:** You must modify the jvm resource pool to match the capabilities of your hardware so that Vertica Pulse has adequate resources to perform queries. If a cluster does not have sufficient resources to run a Vertica Pulse query, then such a query can fail with an Out Of Memory (OOM) exception.

Vertica Pulse runs as a Java UDx (User Defined eXtension) and uses the jvm resource pool to define the resources available to run Vertica Pulse queries.

Vertica starts a Java Virtual Machine (JVM) when you perform a Vertica Pulse query. The session from which you issue the query reserves resources for the JVM (across all nodes in the cluster) and it releases the resources when the session ends. You can also explicitly close the JVM attached to the session by using the command `SELECT release_jvm_memory();`.

The most critical resource pool settings that affect Vertica Pulse are `MAXMEMORYSIZE` and `PLANNEDCONCURRENCY`.

- `MAXMEMORYSIZE` defines the amount of RAM that a JVM can use. By default `MAXMEMORYSIZE` is set to either 10% of system memory or 2GB, whichever is smaller.

- `PLANNEDCONCURRENCY` defines how many JVMs are allowed to run across the cluster and how many Pulse sessions you are able to run cluster-wide. By default, `PLANNEDCONCURRENCY` is set to AUTO, which is the lower of either the number of cores on the node, or memory / 2GB, but it is never automatically set to less than "4".

The amount of memory that each JVM is allocated is determined by `MAXMEMORYSIZE` / `PLANNEDCONCURRENCY`. For example, suppose `MAXMEMORYSIZE` is set to 8G and `PLANNEDCONCURRENCY` is set to 2. In this case, only a maximum of 2 sessions can run Vertica Pulse queries and the session JVMs have a maximum memory size of 4GB.

**Tip:** The basic thing to remember is that `PLANNEDCONCURRENCY` controls the number of sessions across the entire cluster that can run the `sentimentAnalysis()` function. If set to 1, then only a single session can run Pulse functions. No other sessions are able to run Pulse or Java UDx functions until the session currently running Pulse functions is closed.

While resource pool settings are based on the resources of a node, they apply across the entire cluster. A session with a Vertica Pulse query reserves the same resources for its JVM on all nodes in the cluster. Therefore, it doesn't matter if the cluster contains 3 nodes or 30 nodes;
each node reserves, for example, 4GB of the node's memory for the JVM used by the Vertica Pulse session and PLANNEDCONCURRENcy limits the amount of sessions that can run Pulse cluster-wide. If PLANNEDCONCURRENcy is 1, then only 1 vsql session (or client connection) in the entire cluster can run Pulse.

You can display the current resource pool settings for the jvm resource pool with the following command:

```sql
select name, MAXMEMORYSIZE, PLANNEDCONCURRENcy from V_CATALOG.RESOURCE_POOLS
where name = 'jvm';
```

### Configuring the jvm Resource Pool for your System

Do not use the default jvm resource pool settings for Vertica Pulse. You must configure the jvm resource pool to match your hardware and workload requirements. Specifically, specify PLANNEDCONCURRENcy and MAXMEMORYSIZE to match your hardware.

You may need to experiment to find the optimal settings for your hardware and your specific workloads. As a best practice, allow:

- At least 2GB of memory per session for Vertica Pulse
- At least 25% of the memory available for general Vertica overhead. Essentially, MAXMEMORYSIZE must never exceed 75% of total system memory.

**Note:** If you are running a lot of queries not in the context of Vertica Pulse, then you should allow for more memory to be available outside of the jvm resource pool.

To configure your system for Vertica Pulse:

- Determine the number of cores on a node. Your PLANNEDCONCURRENcy setting cannot exceed this value. For example, you can run the following from a shell to determine cores:

  ```bash
  cat /proc/cpuinfo | egrep "core id|physical id" | tr -d "\n" | sed s/physical/\nphysical/g | grep -v "$" | sort | uniq | wc -l
  ```

- Determine the amount of memory in GB on a node. Your MAXMEMORYSIZE cannot exceed 75% of the total system memory. For example, you can run the following from a shell to determine the Total System Memory in GB for any particular node:

  ```bash
  awk /MemTotal/ '{printf "%f GB\n", $2/1024/1024}' /proc/meminfo
  ```
Use the formula $\text{MAXMEMORYSIZE} / \text{PLANNEDCONCURRENCY}$ to determine how much memory each Vertica Pulse JVM receives. For example, you can use $(.75 \times \text{Total System Memory}) / \text{PLANNEDCONCURRENCY}$ if you plan to use most of your RAM for Vertica Pulse. The outcome of the formula must be 2 (which denotes GB) or greater. For example, if you have 8GB of total system memory, and your estimated PLANNEDCONCURRENCY is 3, then the formula results in "2" and is acceptable. However, if you have the same amount of memory and PLANNEDCONCURRENCY is set to 4, then the result of the formula is "1.5", which is below the recommended minimum of 2GB. You can either add more RAM to the system or reduce PLANNEDCONCURRENCY to get the resulting number up to "2".

Finally, alter the jvm resource pool. For example, for a cluster with nodes each having 16GB of memory, and you determine to use up to 75% of the total system memory ($0.75 \times 16\text{GB} = 12\text{GB}$) for Vertica Pulse, then you can set the resource pool as follows:

```
ALTER RESOURCE POOL jvm MAXMEMORYSIZE '12G' PLANNEDCONCURRENCY 3;
```

Note: For evaluation purposes on systems with lower memory, set MAXMEMORYSIZE to 75% and PLANNEDCONCURRENCY to 1: ALTER RESOURCE POOL jvm MAXMEMORYSIZE '75%' PLANNEDCONCURRENCY 1; While these settings are unsupported, they do allow you to run simple Vertica Pulse queries. You may experience Out Of Memory exceptions and slow performance.

For additional details, see:

- ALTER RESOURCE POOL
- Managing Workloads
- Java UDx Resource Management
Assign Users to the pulse_users Role and Allow Access to Pulse Functions

When you install Pulse, the install script creates a pulse schema, which contains the user-dictionary and mapping lists used by Pulse. Initially only administrators can read or edit tables in the pulse schema. To give non-administrator database users access to the pulse schema, you assign the user to the 'pulse_users' role, which has all privileges for the pulse schema. The role is created automatically when you install Pulse.

Note: The default dbadmin user has access to the pulse schema by default. You do not need to add the pulse_users role to the dbadmin account.

Granting users Access to the Pulse Schema

To grant non administrator users access to the tables in the Pulse schema:

1. As the dbadmin, if the user does not exist, create the user with the command: `create user username identified by 'password';`

2. As the dbadmin, if the user does not have access to function in the `public` schema, then grant execute privileges with the command: `GRANT execute ON ALL FUNCTIONS IN SCHEMA public TO username;`

   Note: By default, the Pulse functions are created in the `public` schema.

3. As the dbadmin, grant the pulse_user role to the new user with the command: `grant pulse_users to username;`

4. As the user to which you granted the pulse_user role, set the users role to pulse_users with the command: `set role pulse_users;`

   Note: The user must run the set role command per session to read or edit tables in the pulse schema.
Uninstalling Vertica Pulse

Uninstalling Vertica Pulse on hosts and uninstalling Pulse packages require different procedures.

Uninstall Vertica Pulse on Your Hosts

As the dbadmin, run the uninstall script from the node on which you installed the Pulse package:

```
bash /opt/vertica/packages/pulse/uninstall.sh
```

The uninstall script removes all Pulse functions, but does not remove the pulse schema containing the user-dictionary and mapping tables. To remove all Pulse dictionaries and mappings, including custom dictionaries, include the `-r` parameter

```
bash /opt/vertica/packages/pulse/uninstall.sh -r
```

Uninstall Pulse Packages

To uninstall the Pulse package, on the nodes that have the Pulse package installed, use the appropriate command for your package.

- For RPM packages:

  ```
  # sudo rpm -e vertica-pulse
  ```

- For DEB packages:

  ```
  # sudo dpkg --remove vertica-pulse
  ```

The Pulse schema and associated user-dictionary and mapping tables remain in the database. To remove the Pulse schema and its associated tables, run the following command:
DROP SCHEMA pulse CASCADE
Using Pulse
Dictionaries and Mappings

Pulse contains built-in dictionaries and maps that help determine the sentiment of sentences. You have the option of creating and loading user-defined dictionaries and maps.

Dictionaries and Mappings are loaded across all client sessions and remain in memory even if the database is stopped and started.

Dictionaries

Pulse uses a proprietary system dictionary to help score sentiment. The system dictionary is not visible or modifiable. You can, however, alter the default way that Pulse scores sentiment by modifying user dictionaries. The user dictionaries provide flexibility so that you can tune sentiment scoring for your specific domain. You do not have to modify user dictionaries if Pulse is scoring your data appropriately.

Users can apply dictionaries on a per-user basis. Any number of Pulse users can concurrently apply different sets of dictionaries without conflicts and without disrupting the sessions of other users. Each user can have one dictionary of each type loaded at any given time. If a user does not specify a dictionary of a given type, Pulse uses the default dictionary for that type.

Mappings

Maps are lists of synonyms of one or more words that map to another word. Using maps allows you to analyze text that pertains to the same subject or concept but may use slightly different terminology.

For example, you can map both 'Hewlett Packard' and 'Hewlett-Packard' (with hyphen) to 'HP.' Pulse substitutes the mapped words to the core word when it runs its analysis.

Dictionary and Mapping Tables

User dictionaries and a normalization map for each supported language reside in tables inside the Pulse schema. You can see the contents of the tables with simple queries such as:

```
SELECT * FROM pulse.pos_words_en;
```

Or:
SELECT * FROM pulse.pos_words_es;

There is one table per dictionary/map for each language. The table name has the language abbreviation as a suffix. For example, English tables have the suffix "_en" and Spanish tables have the suffix "_es". By default, the user dictionaries and normalization map are empty. You can modify these tables to tune Pulse to your specific needs. After you modify these tables, you must load the changes into memory.

You can update the user dictionaries and normalization tables at any time. To do so, you must run load functions (see LoadDictionary() and LoadMapping()) to load the values from the tables into memory. Your changes affect sentiment scoring only after you load the new values.

Note: Loading a user dictionary or loading a normalization map overwrites the values in memory with the values from the specified table. You cannot append user dictionaries or the normalization map in memory.

The following dictionary table names provide descriptions of the English user dictionaries. For Pulse versions that support Spanish, the same set of dictionaries with the suffix "_es" is present in the Pulse schema.

<table>
<thead>
<tr>
<th>Dictionary Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>white_list_en</td>
<td>Words that are always marked as an attribute. This list augments the built-in Pulse attribute discovery process. Add words that you always want scored to the white_list user dictionary. For example, such words can include nouns, phrases or business-dependent attributes that are not auto-discovered by Pulse. This list is typically modified to increase the accuracy of sentiment scoring for your domain.</td>
</tr>
<tr>
<td>stop_words_en</td>
<td>Words that are never marked as an attribute. Add words that you do not want scored to the stop_words user dictionary. Use this dictionary to filter out attributes that are not of interest to your analysis. This list is typically modified to increase the accuracy of sentiment scoring for your domain. The stop_words dictionary can only contain nouns and compound nouns. If Pulse does not identify a stop word as a noun, it ignores it.</td>
</tr>
<tr>
<td>pos_words_en</td>
<td>Positive words that can be any type of word or phrase. Words in this list are more likely to carry a positive polarity in general. You can also add exact phrases, such as idioms, to this list.</td>
</tr>
</tbody>
</table>
### Dictionary Table Name

<table>
<thead>
<tr>
<th>Dictionary Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Examples:</strong> <em>adroit, resolve, strong, hit the nail on the head</em></td>
</tr>
</tbody>
</table>
| neg_words_en          | Negative words that can be any type of word or phrase that have a negative connotation. Words in this list are deemed more likely to carry a negative polarity in general. You can also add exact phrases, such as idioms, to this list.  
**Examples:** *abhorrent, butcher, racist, wrath, flash in the pan.* |
| neutral_words_en      | Words that indicate a neutral connotation. Words in this list are scored with a sentiment of 0, meaning not positive or negative. |

The following table shows the tables that describe mapping within Pulse.

<table>
<thead>
<tr>
<th>Mapping Table Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalization_en</td>
<td>A list of word pairs used to map like terms (synonyms). You can use this to correct common misspellings and map them to the correct spelling. This list is frequently modified and is empty by default.</td>
<td></td>
</tr>
</tbody>
</table>
|                    |             | base/synonym:  
- 'hp'/ 'hewlettpackard'  
- 'hp'/ 'Hewlett-Packard'  
- 'Obama'/ 'President Obama'  
- 'Obama'/ 'Barack Obama' |

### Loading Dictionaries and Mappings into Pulse

If you have made changes to the Pulse schema tables, then you must load either the dictionaries, the normalization map, or both. After the changes are loaded, Pulse stores them in memory across all sessions in the cluster. Because Pulse automatically loads the dictionaries and mapping at startup, you do not need to reload them after a database restart or system reboot.
To load an individual user dictionary into memory, use the `LoadDictionary()` function with the appropriate parameter and word list.

- LoadDictionary does not append user-dictionary lists. Instead, it overwrites them. If you load a user dictionary more than once with the same list name, then only the most recent user dictionary is loaded for that list name.

- To load the normalization mapping into memory, use the `LoadMapping()` function with the normalization map.

- If you load a mapping with an incorrect mapName, then the result of LoadMapping() is false and the map is not loaded. LoadMapping() does not append maps. Instead, it overwrites them. If you load a map more than once with the same mapName, then only the most recent mapping is loaded for that mapName.

- If LoadMapping() is successful, Vertica returns a success message from each node in the cluster.

### Automatically Loading Dictionaries and the Normalization Map

For ease of use, Pulse ships with a script to automatically load into memory all of the required user dictionaries and the normalization mapping. This script only exists on the node on which you installed the Pulse RPM/DEB package.

You can run the script from within vsql with the following command:

```
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
```

### Manually Loading Dictionaries and the Normalization Map

If you want to manually load certain user dictionaries or mappings from the Pulse schema tables, run the following command. This example loads the pos_words dictionary. See `LoadDictionary()` for valid values for the `listName` parameter and for multilingual version loading.

**Note:** The following examples use the English dictionaries. For Spanish, replace "_en" with "_es".
1. Add a word to the pos_words dictionary:

```sql
=> INSERT INTO pulse.pos_words_en VALUES('SuperDuper');
=> COMMIT;
```

By default, added words are not case sensitive. "ERROR" produces the same results as "error". You can, however, specify a case setting for a single word using the $Case parameter. For example, to identify "Apple", rather than "apple", you would add the following:

```sql
=> INSERT INTO pulse.white_list_en VALUES('$Case(Apple)');
=> COMMIT;
```

2. Load the updated dictionary into Pulse:

```sql
=> SELECT LoadDictionary(standard USING PARAMETERS
listName='white_list') OVER()
FROM pulse.white_list_en;
```

3. If you change the normalization map, you can load the new normalization values with the following command:

```sql
=> SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS
mapName='normalization') OVER() FROM pulse.normalization_en;
```

After loading, Vertica returns a success message and the number of rows (words or word pairs) loaded.

**Dictionary and Mapping Labels**

You can apply a label to any user-defined dictionary or mapping when you load that object. Labels enable you to perform sentiment analysis against a predetermined set of dictionaries and mappings without having to specify a list of dictionaries. For example, you might have a set of dictionaries labeled "music" and a set labeled "movies." The default user dictionaries automatically have a label of "default."

A single dictionary or mapping can have multiple labels. For example, you might label a white list of artists as both "painters" and "renaissance." You could load the dictionary by loading either label. A label can only apply to one dictionary of each type. For example, you cannot have two dictionaries of stop words that share the same label. If you apply a label to multiple dictionaries of the same type, Pulse uses the most recently applied label.
You can view the labels associated with your current dictionaries using the `GetAllLoadedDictionaries()` function. You can also view the label associated with your current mapping using the `GetLoadedMapping()` function.

**Normalization Map Effect on Results**

Before any of the sentiment analysis functions are run on the text, the normalization map is applied. When a sentiment analysis function is run, Pulse replaces the synonym with the base word. The result of the sentiment analysis function displays the mapped words and not the original text. For example, Pulse maps both 'Hewlett Packard' and 'Hewlett-Packard' (with a hyphen) to 'HP' in the results when the normalization map is populated with those terms.

**Before Mapping**

The following example demonstrates sentiment analysis before mapping:

```sql
=> SELECT SentimentAnalysis('Hewlett-Packard was founded in 1939. Hewlett Packard was started in a garage in Palo Alto California') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hewlett-packard</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>hewlett packard</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>garage</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>palo alto california</td>
<td>0</td>
</tr>
</tbody>
</table>

(4 rows)

**Insert Normalization Values and Load Map**

You can add values to the normalization map using an INSERT statement. The following example demonstrates how to insert normalization values and load the map:

```sql
=> INSERT INTO pulse.normalization_en VALUES('HP', 'Hewlett-Packard');
=> INSERT INTO pulse.normalization_en VALUES('HP', 'Hewlett Packard');
=> COMMIT;
```

```sql
=> SELECT LoadMapping(standard_base, standard_synonym
USING PARAMETERS mapName='normalization') OVER()
FROM pulse.normalization_en;
```

You can also map multiple values to the same term using a $LIST parameter. The following example would map multiple alternate names for the city of Boston to the value 'Boston'.

```sql
INSERT INTO normalization_en Values( 'Boston', '$LIST(BOS,beantown,the hub)';
```
After Mapping

The mapping operation replaces the attributes with their counterparts from the normalization list and displays the base terms:

```sql
=> SELECT SentimentAnalysis('Hewlett-Packard was founded in 1939. Hewlett Packard was started in a garage in Palo Alto California') OVER(PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hp</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>hp</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>garage</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>palo alto california</td>
<td>0</td>
</tr>
</tbody>
</table>

(4 rows)
```

The `CommentAttribute()` function also uses the normalization map and displays the base terms instead of the original text:

```sql
=> SELECT CommentAttributes('Hewlett-Packard was founded in 1939. Hewlett Packard was started in a garage in Palo Alto California') OVER(PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hp</td>
</tr>
<tr>
<td>2</td>
<td>hp</td>
</tr>
<tr>
<td>2</td>
<td>garage</td>
</tr>
<tr>
<td>2</td>
<td>palo alto california</td>
</tr>
</tbody>
</table>

(4 rows)
```

Creating Tables for Custom Dictionary Mappings

The Vertica Pulse package includes all the necessary user dictionary and mappings tables. However, you can create your own tables to store additional user dictionaries or mappings. For example:

```sql
CREATE TABLE my_positive_words(word VARCHAR(64));
```

The following example shows how to create a table, add terms to it, and then load the table as a normalization map:

```sql
=> CREATE TABLE myNormalization(base VARCHAR(64), synonym VARCHAR(64));
=> INSERT INTO myNormalization VALUES('hp', 'Hewlett Packard');
=> INSERT INTO myNormalization VALUES('hp', 'Hewlett-Packard');
=> COMMIT;
=> SELECT LoadMapping(base, synonym USING PARAMETERS
```

...
mapName='normalization') OVER() FROM myNormalization;

After loading, Vertica returns a success message from each node in the cluster.

Using Action Patterns in Dictionaries

Vertica Pulse supports the use of action patterns in white_list dictionaries only. An action pattern enables Pulse to recognize phrases that denote action, intention, or interest, such as going to buy, waiting to see, and so on. Action patterns can identify behaviors associated with your sentiment analysis terms.

Action patterns can:

- **Connect Word Forms to a Root Word** — Vertica Pulse lemmatizes all words. *Lemmatization* recognizes different word forms and maps them to the root word. For example, Pulse would map bought and buying to buy. This ability extends to misspellings. For example, tryiiiing and seeeeeing taablets would map to trying and seeing tablets.

- **Create Object-Specific Queries** — To identify only the attributes that are objects of action patterns, create a whitelist dictionary that contains only action patterns of interest. In your sentiment analysis query set the actionPattern and whiteListOnly parameters to true.

  **Note:** Action patterns exist in the whitelist dictionary. If a word that matches an action pattern appears in both the white_list and stop_words dictionaries, the white_list takes precedence. The stop_list word would appear in sentiment analysis results.

Action Pattern Syntax

Construct an action pattern by combining action parameters within an #action. By default, parameters match any instance of the associated part of speech. You can match specific terms by listing them with the parameter. For example, the parameter $PREP(to, on), would match only to and on.

Parameters can also accept $regex and $list operators.

```
#action{$ADV $VERB $PREP $ADJ}
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ADJECTIVE</td>
<td>$ADJ</td>
<td>Matches any adjective.</td>
</tr>
</tbody>
</table>
Matches any adverb.

Matches any preposition.

Matches any verb.

Default Action Patterns

Pulse includes default action patterns in the whitelist dictionary. You cannot remove these patterns. Pulse always evaluates them when you perform a sentiment analysis.

<table>
<thead>
<tr>
<th>Language</th>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>$Verb $Prep $Verb</td>
<td>planning on buying, thinking about dropping</td>
</tr>
<tr>
<td></td>
<td>$Verb TO $Verb</td>
<td>going to buy, looking to acquire</td>
</tr>
<tr>
<td>Spanish</td>
<td>$Verb $Prep $Verb</td>
<td>voy a comprar, pienso en dejar</td>
</tr>
<tr>
<td></td>
<td>$Verb $Verb</td>
<td>quiero solicitar, planean adquirir</td>
</tr>
</tbody>
</table>

Examples

The following example shows how Pulse can match customer or client, the verb would and any other verb. It would match phrases like customer would buy or client would cancel.

```
INSERT INTO pulse.white_list_en values('#action($LIST(customer,client) would $VERB}');
```

The following example shows a match for specific verbs like, want, and plan, plus any preposition and any other verb. It would match phrases like want to own or plan on buying.

```
INSERT INTO pulse.white_list_en values('#action($VERB(like,want,plan) $PREP $VERB}');
```

The following example identifies words ending in ember, such as December, and uses a regular expression to identify date references, such as 2nd or 4th. This action pattern could identify users planning to attend an event or making holiday plans.

```
INSERT INTO pulse.white_list_en values('#action(On $regex(.+ember) $regex(\d+(th|st|rd|nd)) I will $verb to}');
```
Using Lists In Dictionaries

Vertica Pulse supports the use of token-based lists in dictionaries. You can use a list to match multiple terms to a single word. Token based lists differ from mapping by allowing you to create multiple associations in a single action rather than a series of pairs. Unlike mapping, lists are not restricted to the normalization dictionary.

**Note:** Token-based lists do not apply to the base word in normalization dictionaries.

You can add lists to the following dictionaries:

- `pos_words`
- `neg_words`
- `neutral_words`
- `normalization`
- `white_list`
- `stop_words`

You can add a token to a user-defined dictionary using an INSERT statement and `$LIST` parameter containing the list of values to match.

The following example would match slang terms to the word "good".

```
INSERT INTO pos_words Values( 'good', '$LIST(sweet,dope,tight)');
```

Using Regular Expressions in Dictionaries

Vertica Pulse supports the use of regular expressions in user-defined dictionaries. Vertica Pulse regular expressions use the `java.util.regex` package syntax. For more information on this syntax, refer to the Oracle documentation.

**Note:** Regular expressions do not apply to the base word in normalization dictionaries.

You can add regular expressions to the following dictionaries:
- pos_words
- neg_words
- neutral_words
- normalization
- white_list
- stop_words

You can add a regular expression to a user-defined dictionary using an `INSERT` statement and `$REGEX` parameter containing the regular expression. Regular expressions are case insensitive. the regular expression `$regex(apple)` produces the same matches as the regular expression `$regex(Apple)`.

Note: A regular expression can support a single token or word. *Smartphone* would be a valid token, but *smart phone* would not.

The following example would match any word ending with the string "day". You could use it to identify the days of the week or words such as *yesterday* and *today*.

```
INSERT INTO stopwords_en Values('$LIST(nice,good,fine) $REGEX(.*day)');
```

The following example matches references to iPhones, including the number and letter version.

```
INSERT INTO whitelist_en Values('Iphone $REGEX(\d{1}(S|C))?');
```

To use a parenthesis as a literal part of a regular expression, you must use the escape character `\` twice to prevent Pulse from interpreting the parenthesis as metacharacter in the regular expression. The following example would match the literal string (*hugs*)

```
INSERT INTO whitelist_es Values($REGEX(\\(hugs\\)));
```
**Determining Sentiment**

You determine sentiment by using the `SentimentAnalysis()` function on text.

The `SentimentAnalysis()` function first extracts the attributes (typically nouns) from the sentence, and then applies a sentiment score to each attribute. Scores can be one of the following:

- **1** - Positive Sentiment
- **0** - Neutral Sentiment
- **-1** - Negative Sentiment

This provides a more granular analysis than just determining the sentiment for the sentence as a whole. Consider the following quote from Abraham Lincoln; "Force is all-conquering, but its victories are short-lived." If you were to score the sentiment of the sentence as a whole by averaging the sentiment of its parts, then the sentiment is neutral.

```sql
=> select avg(t1.sentiment_score) as 'Average Sentiment' from (  
    select sentimentAnalysis('Force is all-conquering, but its victories are short-lived.')  
    over (PARTITION BEST)  
) as t1;

Average Sentiment
-----
0
```

If you score the individual attributes of the sentence, then you can obtain a much more precise analysis of the sentiment than if you were trying to assign a single score to the entire sentence. For example:

```sql
=> select sentimentAnalysis('Force is all-conquering, but its victories are short-lived.') over (PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Force is all-conquering, but its victories are short-lived.'</td>
<td>force</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>victories</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

"Force" is scored with positive sentiment because it is "all-conquering". "Victories" is scored with negative sentiment because it is "short-lived".

**Note:** Vertica Pulse does not recognize personal pronouns (I, you, we, he, she, it, etc.) as attributes.
SentimentAnalysis() also extracts the sentiment from multiple sentences and returns the sentence in which attributes are found:

```sql
=> SELECT SentimentAnalysis('Force is all-conquering, but its victories are short-lived. Every good boy deserves fudge.') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>force</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>victories</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>boy</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>fudge</td>
<td>1</td>
</tr>
</tbody>
</table>

(4 rows)

"Boy" is scored with positive sentiment because he is good. Fudge is scored with positive sentiment because it is something that is deserved.

**Note:** The sentence detector considers a period to mark the end of a sentence. Some abbreviations that use a period, such as Dr. or Mr., cause the sentence detector to end the sentence at the abbreviation.

The SentimentAnalysis function also identifies attributes with neutral sentiment (a sentiment score of zero). For example:

```sql
SELECT SentimentAnalysis('Roses are red. Violets are blue.') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>roses</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>violets</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

Both roses and violets receive neutral sentiment because neither being red nor blue is considered positive or negative in this context.

See the Pulse Cookbook for more examples of determining sentiment.

**Sentiment Analysis Levels**

Vertica Pulse is capable of determining sentiment at the following levels:

- Attribute
- Sentence
- Document

You can specify an analysis level using the granularity parameter of the SentimentAnalysis function. You can perform multiple levels of analysis simultaneously within the same query.
Attribute-Level Analysis

Attribute level analysis provides a sentiment for each object in a sentence. This behavior is the default level of analysis for Pulse. Attribute analysis identifies the objects of a sentence and any sentiment expressed regarding those objects.

The following example shows the sentiment expressed with regard to "camera" and "quality pictures."

```
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one') USING PARAMETERS granularity='A' over();
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>camera</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>quality pictures</td>
<td>1</td>
</tr>
</tbody>
</table>

Sentence-Level Analysis

A sentence level analysis provides the overall sentiment of each sentence in a document. If a sentence is contains both positive and negative sentiments, it appears as mixed.

The following example shows two sentences, the first of which is mixed. As a mixed sentiment, the sentiment score is 0, or neutral, and the mixed value is true. The second sentence is entirely positive. Its sentiment is 1, or positive, and the mixed value is false.

```
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one') USING PARAMETERS granularity='S' over();
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>sentiment_score</th>
<th>mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>false</td>
</tr>
</tbody>
</table>

Document-Level Analysis

Document level analysis provides the overall sentiment of an entire document. If you wanted to know if a movie review was positive, negative, or mixed, a document level analysis could provide that information. Document level analysis gives both the overall sentiment score and a mixed rating if the sentiment is not exclusively positive or negative.

The following example shows that overall, the writer is positive but does express some negative sentiments.
Select SentimentAnalysis ('The camera takes great quality pictures but is expensive. It feels like a professional one' USING PARAMETERS granularity='D') over();

<table>
<thead>
<tr>
<th>sentiment_score</th>
<th>mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>
Tuning Pulse

Pulse contains built-in dictionaries that help to determine the sentiment of sentences. These dictionaries are not directly readable. However, you can modify the Pulse dictionary tables to improve automatic attribute discovery and provide more accurate results for sentiment scoring based on your specific data sets. The dictionary tables are available in the Pulse schema. Any words you add to these dictionaries takes precedence over the built-in dictionaries.

Improving Automatic Attribute Discovery

Pulse identifies nouns in sentences and marks them as attributes. However, there are two dictionaries and one mapping that you can modify to improve automatic attribute discovery. These are:

- **white_list** - a list of words on which you want to score sentiment, but are not auto-discovered by Pulse. Typically these are product or company names, or special words in the domain of the data you are analyzing. You can also add noun phrases to the white_list.

- Consider the term "President Smith". Pulse automatically marks "President" as an attribute. However, you can add "President Smith" to the white_list and Pulse then uses "President Smith" as the attribute instead of just "President".

- If your white_list contains phrases that are subsets of other phrases in the white list, then the shorter phrase is not matched if the text being analyzed matches the superset phrase. For example, if both "Honest Al" and "Honest Al Car Emporium" are in the white_list, then the latter phrase is identified as an attribute in the text "Honest Al Car Emporium is not honest.", not the shorter "Honest Al" white_list phrase.

- **stop_words** - a list of words on which you do not want to score sentiment, but may appear frequently in your data set. stop_words is basically a way to filter out attributes.

- If a word appears in both stop_words and white_list, then the white_list word takes precedence. The word appears in results even though it is in the stop_words dictionary.

- normalization - a map of base words and synonyms that allow you to normalize words for easy comparison. For example, you can normalize "Hewlett Packard" to "HP", then count the number of times "HP" appears as an attribute in your data. Any text that contains "HP" or "Hewlett Packard" is counted towards the total.
Determining How Pulse Scores Sentiment

When tuning Pulse it is important to understand why Pulse may not be scoring a particular attribute the way you want it to be scored. For example, consider the sentence "The quick brown fox jumped over the lazy dog." By default, Pulse scores the fox as positive and the dog as negative. If you want to better understand how the words in the sentence affect the attributes, then you can use the relatedwords parameter to see which words are affecting the score. For example:

```
select SentimentAnalysis('The quick brown fox jumped over the lazy dog.' USING PARAMETERS relatedwords=true) OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
<th>related_word_1</th>
<th>related_word_2</th>
<th>related_word_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
<td></td>
<td>1</td>
<td>quick</td>
<td>lazy</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
<td>-1</td>
<td>lazy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

The output details that "quick" and "lazy" impacted the scoring of the "fox" attribute, and that "lazy" affected the scoring of the "dog" attribute. "Quick" (positive) is weighted heavier than "lazy" (negative) when scoring "fox" because the word "quick" is closer to the attribute "fox" in the sentence, and the result is that "fox" is scored positively. "Lazy" (negative) is the only related word being used to score the sentiment for "dog". If you don't agree with the scoring, you can change how these related words affect the score by adding them to the appropriate user-dictionary, as described in "Improving Sentiment Scores".

Improving Sentiment Scores

Pulse scores sentiment on attributes (nouns) in sentences using Natural Language Processing (NLP) algorithms and rules. Pulse attempts to identify the parts of a sentence (for example, verbs, nouns/attributes, adjectives, etc; the parts of speech), and then scores the attributes based on which system-dictionaries the parts of speech appear (positive, negative, or neutral) and where those parts of speech appear in relation to the attributes and other contextual information. Pulse does not identify personal pronouns (he, you, we, she, etc.) as attributes.

Pulse provides a PartsOfSpeech function so that you can verify which parts of speech are being identified in a sentence.
Sentiment Scoring and the Precedence of Pulse User-Dictionaries

The negative, positive, and neutral user-dictionaries adjust the score of an attribute based on which dictionary the words in the sentence appear. User-dictionaries take precedence over the internal dictionaries that Pulse uses for analyzing text, so that you can override the default polarity of an opinion word or phrase by inserting it in the appropriate user-dictionary table.

Pulse also supports using phrases in the pos_words, neg_words and neutral_words dictionaries. Phrases, such as idioms ("hit the nail on the head."), can be added to the user dictionaries. Phrases of two or more words support "fuzzy" matching. For example, the phrase "solve problem" also matches "solves problems".

Pulse uses an order of precedence to determine which user dictionary is used to modify the default score. The order of precedence of the user dictionary that Pulse uses to score attributes is as follows:

1. Phrases or strings that occur in the "neutral_words" dictionary
2. Phrases or strings that occur in the "neg_words" dictionary
3. Phrases or strings that occur in the "pos_words" dictionary
4. Single words appearing in the "neutral_words" dictionary
5. Single words appearing in the "neg_words" dictionary
6. Single words appearing in the "pos_words" dictionary

Note: If a word is present in both stop_words and white_list, then the white_list word takes precedence. The word is present in results even though it exists in stop_words.

Consider the sentence "Fudge is good". It contains three parts; a noun (fudge), a verb (is), and an adjective (good). When you analyze the sentence using Pulse, it identifies "fudge" as an attribute, because it is a proper noun, and then assigns "fudge" a positive sentiment:

```sql
select sentimentAnalysis('Fudge is good') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fudge is good</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The number of words matched against a dictionary also has an impact on which dictionaries take precedence over single words. For example, the positive phrase "solve problem" causes a positive score on the text "Joe solves problems", even though "problem" is a negative word. Since phrases have precedence over single words, a positive score is applied to Joe.

```
SELECT SentimentAnalysis('Joe solves problems.') OVER(PARTITION BEST);
```

```
<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>joe</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)
```

```
SELECT SentimentAnalysis('Joe is a problem.') OVER(PARTITION BEST);
```

```
<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>joe</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>problem</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)
```

**Tuning Example**

You can modify any of the user-dictionaries to improve the accuracy of sentiment scores. The two basic dictionaries, "neg_words" and "pos_words", are typically the easiest to modify to get good results. Words in these two dictionaries can be any part of speech (verb, adjective, etc.). If you find a word that is causing an attribute to be scored positively or negatively, but it should be scored as neutral, then you can add that word to the "neutral_words_en" dictionary to cause it to be scored 0.

Consider the sentence "The product delivers simplicity."

```
select sentimentAnalysis('The product delivers simplicity.') over(PARTITION BEST);
```

```
<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>product</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>simplicity</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)
```

If you want "product" to be scored positively in this sentence, then you must add "deliver simplicity" to the pos_words user-dictionary. "deliver simplicity" will also match "delivers simplicity" due to the "fuzzy" matching feature of phrases in the dictionaries. If you add "simplicity" by itself to the "pos_words" dictionary, then simplicity in any context is considered positive, which may not be the result you want to achieve across your entire domain. The following example adds "deliver simplicity" to the pos_words user-dictionary for the English language:

```
insert into pulse.pos_words_en values ('deliver simplicity');
commit;

-- you must reload the dictionaries for the changes to be effective
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql

```
select sentimentAnalysis('The product delivers simplicity.') over(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>product</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)

Notice that "simplicity" is not positive if it is not paired with "deliver":

```
select sentimentAnalysis('The product provides simplicity.') over(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>product</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>simplicity</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

If you want "simplicity" to always be positive, add it to the "pos_words" list. This example replaces "deliver" with "provides":

```
insert into pulse.pos_words_en values ('simplicity');
commit;
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql

select sentimentAnalysis('The product provides simplicity.') over(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>product</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>simplicity</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

Notice that the sentiment score for the attribute (noun) "simplicity" is not affected by having the word "simplicity" in a Pulse user-dictionary, since it has been identified as an attribute.

**Additional Tuning Examples**

The following table provides additional examples for tuning Pulse.

<table>
<thead>
<tr>
<th>Text</th>
<th>Attribute</th>
<th>Score</th>
<th>Tuning Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>New product smashes kickstarter target in a day!</td>
<td>New Product</td>
<td>Default: -1</td>
<td>&quot;Smash&quot; is scored negatively by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After</td>
<td>default. Add &quot;smash&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tuning: 1</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>Attribute</td>
<td>Score</td>
<td>Tuning Steps</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Get a sneak peek of the new movie.</td>
<td>Movie</td>
<td>Default: -1</td>
<td>&quot;sneak&quot; is scored negatively by default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After Tuning: 1</td>
<td>Add &quot;sneak peek&quot; to &quot;pos_words&quot;.</td>
</tr>
<tr>
<td>Google was able to spot trends in flu outbreaks in the United States using the collection and analysis of big data.</td>
<td>Google</td>
<td>Default: -1</td>
<td>&quot;outbreak&quot; is scored negatively by default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After Tuning: 1</td>
<td>Add &quot;spot trend&quot; to &quot;pos_words&quot;.</td>
</tr>
<tr>
<td>Five health tips that will knock your socks off!</td>
<td>health tips</td>
<td>Default: -1</td>
<td>&quot;knock&quot; is scored negatively by default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After tuning: 1</td>
<td>Add &quot;knock your socks off&quot; to &quot;pos_words&quot;.</td>
</tr>
</tbody>
</table>

If you have many words or base/synonyms to add to user-dictionaries, then you can bulk load the lists from text files. See Bulk Loading Word Lists from Text Files.

**Bulk Loading Word Lists from Text Files**

If you have many words that you need to add to the user-dictionary or normalization mapping, it may be easier to create the word lists in a text file and load the lists using the COPY command.
Bulk Loading User Dictionary Lists

To bulk load user-dictionary lists into the pulse schema, first create a text file with the list of words to add, one word per line, for each of the user-dictionaries. See Dictionaries and Mappings for a list of the user-dictionaries and normalization map. Optionally name each text file to match the name of the corresponding user-dictionary. Place these text files in the /home/dbadmin directory.

Then, in vsql, use one or more of the following commands to load the respective text file into the pulse schema. These commands assume that you are using English version of Pulse, that the built-in user dictionary tables in the pulse schema and that the text files are named the same as the user-dictionary.

```sql
COPY pulse.neg_words_en(standard) from '/home/dbadmin/neg_words.txt';
COPY pulse.neutral_words_en(standard) from '/home/dbadmin/neutral_words.txt';
COPY pulse.pos_words_en(standard) from '/home/dbadmin/positive_words.txt';
COPY pulse.stop_words_en(standard) from '/home/dbadmin/stop_words.txt';
COPY pulse.white_list_en(standard) from '/home/dbadmin/white_list.txt';
```

Bulk Loading the Normalization Map

You can load normalization terms into the pulse schema similarly to loading user-dictionaries. However, instead of one word per line, use the convention of one pair of words per line, separated by a comma. For example, to map the different forms of DC (District of Columbia), create a text file in /home/dbadmin named normalization.txt with the following content:

```
DC, District of Columbia
DC, D.C.
```

Then, in vsql, use the following command to load the normalization into the pulse schema.

```bash
$ COPY pulse.normalization_en (standard_base, standard_synonym) FROM '/home/dbadmin/normalization.txt' DELIMITER ',';
```

When you have finished loading the text files, run the loadUserDictionaries.sql script to update the new terms in memory:

```bash
vsql -f /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
```
**Multilingual Pulse**

This section describes the multilingual features of Pulse and gives a brief explanation on how to use the sentimentAnalysis() functions for different supported languages.

Pulse can analyze text in different languages. Currently English and Spanish are supported. You can specify the language that is analyzed in three ways:

- **Provide the language as argument:** if there is a language specified in the document record, then it can be used for analyzing the text by passing it as argument. This is particularly useful when a dataset contains texts in different languages. If the language in a record is not a supported one, then it is ignored.

- **Provide the language as parameter:** if there is no value specified for the language for a document record, Pulse uses the value specified for the language parameter in the query to get the language.

  **Note:** If you provide the language parameter more than once, then the last value specified is used.

- **Do not provide an argument or parameter and use the default language:** If the language is neither specified in the record nor by the user, then Pulse defaults to English unless you have changed the default language. To change the default language use the `SetDefaultLanguage` function.

  **Note:** If you provide both an argument and a parameter, then the argument is used as the language. If the argument is not valid then the parameter is used. If neither the argument or parameter are valid then the default language is used.

  **Note:** Accents are removed from characters in attributes. Additionally, a "u" with a dieresis is converted to a plain "u" and an "n" with a diacritical tilde is replace with a plain "n".

Functions that use language as parameter and/or as argument:

- **CommentAttributes**
- **ExtractSentence**
- **GetAllSentences**
- **GetSentenceCount**
- PartsOfSpeech
- SentimentAnalysis

Other functions can use the language only as a parameter (if not provided, the function uses the default language):

- GetLoadedDictionary
- GetLoadedMapping
- LoadDictionary
- LoadMapping
- GetAllDictionaryWords
- GetAllMappingWords

**In This Section**

**Spanish Pulse**

The only visible difference between the English and Spanish versions is in the table names for the user dictionaries. The modifications for dictionaries/mappings must be done in the following tables:

- white_list_es
- stop_words_es
- pos_words_es
- neg_words_es
- neutral_words_es
- normalization_es
Consider the text "El producto provee simplicidad" (the product provides simplicity). If the word 'simplicidad' (simplicity) should be positive, it has to be loaded into the pos_words dictionary for Spanish as follows:

```sql
select sentimentanalysis('El producto provee simplicidad') OVER(PARTITION BEST);
```

```
+----------------+-----------------+-----------------+
| sentence | attribute | sentiment_score |
|----------+-----------+-----------------|
| 1        | producto  | 0               |
| 1        | simplicidad | 0             |
```

(2 rows)

```sql
insert into pulse.pos_words_es values('simplicidad');
```

(1 row)

```sql
select LoadDictionary(standard USING PARAMETERS listName='pos_words') over() from pulse.pos_words_es;
```

Success

```
1
```

(1 row)

```sql
select sentimentanalysis('El producto provee simplicidad') OVER(PARTITION BEST);
```

```
+----------------+----------------+-----------------+
| sentence | attribute | sentiment_score |
|----------+-----------+-----------------|
| 1        | producto  | 1               |
| 1        | simplicidad | 0             |
```

(2 rows)

**Multilingual Examples**

**Language as an Argument**

```sql
select sentimentanalysis('Cookies are sweet.', 'english') OVER(PARTITION BEST);
```

```
+----------------+----------------+-----------------+
| sentence | attribute | sentiment_score |
|----------+-----------+-----------------|
| 1        | cookies   | 1               |
```

(1 row)

```sql
select sentimentanalysis('Las galletas son dulces','spanish') OVER(PARTITION BEST);
```

```
+----------------+----------------+-----------------+
| sentence | attribute | sentiment_score |
|----------+-----------+-----------------|
| 1        | galletas  | 1               |
```

(1 row)

The following example shows how to analyze tweets from a table where each tweet record contains the language of the tweet in addition to the text.
create table myTweets (text varchar(300), language varchar(15));

insert into myTweets values ('Wired reviews Amazon’’s tiny-screen Kindle Fire: Web browsing sucks, emotionally draining, makes reading a chore’, 'english');

insert into myTweets values ('Cookies are sweet', 'english');

insert into myTweets values ('Why does my iPhone have 6 GB of corrupted space I can’’t use? That is obnoxious.’, 'english');

insert into myTweets values ('Las galletas son dulces’, 'spanish');

insert into myTweets values ('el iPhone es el celular mas popular’, 'spanish');

select sentimentanalysis(text,language) OVER(PARTITION BEST) from MyTweets;

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
</table>
| reviews amazon            | -1
| kindle fire               | -1
| web                       | -1
| chore                     | -1
| cookies                   | 1
| iphone                    | -1
| gb                        | -1
| space                     | -1
| galletas                  | 1
| iphone                    | 1
| celular                   | 1

(11 rows)

Language as a Parameter

select sentimentanalysis('Las galletas son dulces’ using PARAMETERS language='spanish') OVER(PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>galletas</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

select sentimentanalysis('Cookies are sweet' using PARAMETERS language='english') OVER(PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookies</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

Although it is possible to specify the language as parameter for a specific text given in a query, using the language argument is more appropriate. The use of the language parameter is targeted to queries that analyze a set of texts (from a table) written in a same language. The language parameter is used by Pulse to skip texts in other languages because Pulse does not automatically detect the language, Thus, Pulse uses the language specified as parameter to
analyze each text from the table (consequently the sentiment scores for texts in other language may be incorrect).

The following example shows a query that analyzes tweets from a table where the tweets do not have a language value stored in the table, but are all in the same language.

```
create table myTweets (text varchar(300));
insert into myTweets values ('Las galletas son dulces');
insert into myTweets values ('el iphone es el celular mas popular');
insert into myTweets values ('el zorro rapido brinco sobre el perro flojo');
select sentimentanalysis(text using PARAMETERS language='spanish') OVER(PARTITION BEST) from MyTweets;
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>galletas</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>iphone</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>celular</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>zorro</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
<td>-1</td>
</tr>
</tbody>
</table>

(5 rows)

The following example shows a query that analyzes tweets from a table with tweets in different languages. The Spanish tweets do not have the language value. In a single query you can specify both an argument and parameter. The argument has precedence over the parameter setting. In this case the parameter is only used when a tweet doesn't provide a language value.

```
create table myTweets (doc_id int, text varchar(300), language varchar(15));
insert into myTweets values (1, 'Vertica is the best company', 'english');
insert into myTweets values (2, 'Cookies are sweet', 'english');
insert into myTweets values (3, 'The quick brown fox jumped over the lazy dog', 'english');
insert into myTweets values (4, 'Las galletas son dulces');
insert into myTweets values (5, 'el iphone es el celular mas popular');
select doc_id, sentimentanalysis(text,language using PARAMETERS language='spanish') OVER(PARTITION BY id, text) from MyTweets;
```

<table>
<thead>
<tr>
<th>doc_id</th>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>vertica</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>company</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>cookies</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>fox</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dog</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>galletas</td>
<td>1</td>
</tr>
</tbody>
</table>
Using the Default Language

```
select sentimentanalysis('Cookies are sweet') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cookies</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)
Pulse Cookbook

This section contains the following recipes for using Pulse

Batch Analyzing Data as It Is Loaded

If you are constantly loading data that needs to be analyzed with Pulse, then you should run the sentimentAnalysis() function in batches on the newly loaded data. You can store the sentiment scores in a separate table and associate the rows in the scored table with the original table by joining on IDs between the tables. Running sentimentAnalysis() as the data is loaded and storing the results is more efficient than running sentimentAnalysis() during interactive sessions because the sentimentAnalysis() can take a few seconds to return results.

For example, suppose that you are using the Social Media Connector (available in the ETL and Data Ingest section of the Vertica Marketplace) to retrieve Twitter tweets and load them into Vertica. In this case, you can create shell scripts and a cron job to automatically run sentimentAnalysis() on the text of the tweets. Then you can store the resulting scores in a table for quick retrieval later on.

Complete the following steps as the dbadmin user to run sentimentAnalysis() on your Twitter data. This task also sets up the system to run sentimentAnalysis() on new Twitter data every 2 minutes.

1. Create a table to hold the tweets (for example, named tweets) with the following structure:

   ```sql
   create table tweets(
   id int,
   created_at timezonetz,
   "user.name" varchar(144),
   "user.screen_name" varchar(144),
   text varchar(500),
   "retweeted_status.retweet_count" int,
   "retweeted_status.id" int,
   "retweeted_status.favorite_count" int,
   "user.location" varchar(144),
   "coordinates.coordinates.0" float,
   "coordinates.coordinates.1" float,
   lang varchar(5)
   );
   ```
The columns are based on the data returned by Twitter's streaming API. The fields are defined in the Twitter Field Guide at [https://dev.twitter.com/docs/platform>/tweets](https://dev.twitter.com/docs/platform>/tweets).

Note that the columns with quoted names; "user.name", "user.screen_name", are sub-fields within a larger field. For example, the "users" field is described here: [https://dev.twitter.com/docs/platform>/users](https://dev.twitter.com/docs/platform>/users).

You must at least have columns for id, text, and "user.screen_name"

2. Create a table to hold the sentiment scores (for example, named : `tweet_sentiment`). Then load it with the scores from your existing tweets. Make sure no new tweets are loaded until this step completes.

Replace the column names in the following example with the column names from your twitter table. The example uses the column names used by the Social Media Connector:

```sql
create table tweet_sentiment as
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true)
over (partition by id, "user.screen_name", text)
from tweets where lang='en' order by attribute);
```

---

Note: If you have a large number of tweets then this command can take a long time to run. However, it is important to score your existing data before you start scoring newly loaded data.

3. Create a SQL script to update the `tweet_sentiment` table with data from newly loaded tweets. Save it in the home folder of the Vertica database admin user. For example, this path could be `/home/dbadmin/tweet_update.sql`.

Replace the column names with the column names from your twitter table. The following example uses the column names used by the Vertica Social Media Connector:

```sql
\i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
drop table if exists dt_end;
create table dt_end as (select max(created_at) dt from tweets);

-- run sentiment
insert into tweet_sentiment
(select id, "user.screen_name",
SentimentAnalysis(text using parameters filterlinks=true,
filterusermentions=true)
```
over (partition by id, "user.screen_name", text)
from tweets where lang='en' and
tweets.created_at > (select dt from dt_start) and
tweets.created_at <= (select dt from dt_end)
order by attribute);

-- copy date end into new start date
drop table if exists dt_start;
create table dt_start as (select dt from dt_end);

-- free up jvm resource pool memory used by this script
select release_jvm_memory();

4. Create a shell script named `tweet_update.sh` that is run from a cron job. This shell script runs the `tweet_update.sql` script and logs the results to the file `tweet_update.log`. Save the `tweet_update.sh` script in the home folder of the Vertica database admin user. For example, this path could be `/home/dbadmin/tweet_update.sh`.

Replace the `dbadmin`, `password`, and `databasename` values with the values for your system.

```
/opt/vertica/bin/vsql -U dbadmin -w password -d databasename -f /home/dbadmin/tweet_update.sql > tweet_update.log
```

After you have created the shell script `tweet_update.sh`, make the script executable by entering the following command: `chmod +x tweet_update.sh`.

5. Create a cron job to run the script every two minutes. Use the command `crontab -e` to create the cron job. You can view all of your created cron jobs by using the command `crontab -l`.

```
*/2 * * * * /home/dbadmin/tweet_update.sh
```

The script runs every two minutes. Any new tweets that have been loaded in that two-minute window are analyzed and the results are added to the `tweet_sentiment` table. You can join results of queries by the id's of the `tweets` and `tweet_sentiment` tables.
Analyzing Comments for a Company or Product

Pulse allows you to analyze comments (such as tweets) for a particular company or product.

For example, imagine that the fictional company Pytell Corp has just released a new product called Owl-2. You want to analyze the sentiment of both the company and the product.

You've collected several tweets from Twitter about several companies and products into your database. However, for this analysis you only want to target tweets that have to do with Pytell Corp and/or Owl-2.

The dataset for this example is below:

```
create table tweets_sample(id int, author varchar(50), text varchar(400));

insert into tweets_sample values(400900, 'DramaBugs', 'Pytell Corp has horrible customer support. On Hold 2 hours!');
insert into tweets_sample values(401200, 'Gemball', 'Owl-2 doesn''t fly!');
insert into tweets_sample values(403070, 'Postta', 'Pytell finally released Owl-2!');
insert into tweets_sample values(480920, 'Instana', 'Unboxing Owl-2 after work today! Stay Tuned!');
insert into tweets_sample values(434500, 'Dailydant', 'Owl-2 flies great! I like it!');
insert into tweets_sample values(450670, 'HelpfulBen', 'Owl-2 keeps crashing into things!');
insert into tweets_sample values(402092, 'Championtips', 'Owl-2 has solved our rodent infestation!');
insert into tweets_sample values(434950, 'Editone', 'Pytell fail? Reports of Owl-2 crashing through windows.');
insert into tweets_sample values(403511, 'BuffDrama', 'I am afraid of small spiders.');
commit;
```

1. Run `SentimentAnalysis` to get an idea of how Pulse is analyzing the data:

```
SELECT author, SentimentAnalysis(text) OVER(PARTITION BY author, text) FROM tweets_sample ORDER BY attribute;
```

<table>
<thead>
<tr>
<th>author</th>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
</table>
2. There are some attributes listed (ponies!) that do not apply to the analysis that you are doing. You can focus your analysis by adding whitelist entries and filtering on the whitelist. Insert whitelist entries for the company and product name into the standard whitelist:

```sql
INSERT INTO pulse.white_list_en VALUES ('Pytell Corp');
INSERT INTO pulse.white_list_en VALUES ('owl-2');
commit;
```

Reload the whitelist into Pulse. Loading a user-dictionary or mapping overwrites the existing user-dictionary or mapping:

```sql
SELECT LoadDictionary(standard USING PARAMETERS listName='white_list') OVER() FROM pulse.white_list_en;
```

3. Also, note that Pulse is not identifying all variations on the company name. There are also three obvious attributes for the product name ('Pytell', 'pytell corp'). You can normalize these values by using a normalization mapping. Add the synonyms to the standard normalization mapping:

```sql
insert into pulse.normalization_en values('Pytell', 'Pytell Corp');
commit;
```

4. Reload the normalization mapping to load the new values into Pulse:
5. Run the query again to see how the normalization affects the results.

   Note that 'pytell corp' has been normalized to 'pytell' and Pulse is correctly identifying the synonyms and mapping them to the base term.
Determining Popular Topics

The next examples in this cookbook use a table with the following structure:

```sql
create table tweets(
    id int,
    created_at timezonetz,
    "user.name" varchar(144),
    "user.screen_name" varchar(144),
    text varchar(500),
    "retweeted_status.retweet_count" int,
    "retweeted_status.id" int,
    "retweeted_status.favorite_count" int,
    "user.location" varchar(144),
    "coordinates.coordinates.0" float,
    "coordinates.coordinates.1" float,
    lang varchar(5)
);
```

The columns are based on the data returned by Twitter's streaming API. The fields are defined in the Twitter Field Guide at https://dev.twitter.com/docs/platform>/tweets.

Note that the columns with quoted names; "user.name", "user.screen_name", are sub-fields within a larger field. For example, the "users" field is described here: https://dev.twitter.com/docs/platform>/users.

The example queries provided work with any Twitter data that follows the above table structure.

---

Determining Popular Topics

The Pulse attribute discovery feature allows you to easily find popular topics in a data set. Use the `CommentAttributes()` function to extract the attributes from rows of text and count the number of times the attribute occurs.

For example, using a dataset of 30,000 tweets that matched a keyword of "D11" collected during the D11 tech conference in 2013, you could get a count of the attributes discovered by Pulse to determine popular topics:

```sql
SELECT t.attribute, count(*) FROM(SELECT CommentAttributes(text)
OVER(PARTITION BEST) FROM tweets) as t
GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

<table>
<thead>
<tr>
<th>attribute</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If the dataset contains tweets in English and Spanish languages, then (using the Pulse multilingual version) each tweet can be analyzed according to its language by specifying the language as argument in the CommentAttributes() function. If the language of a specific tweet is not supported, then that tweet is ignored by the function. For example:

```sql
SELECT t.attribute, count(*) FROM (SELECT CommentAttributes(text,lang) OVER(PARTITION BEST) FROM tweets) as t GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

Notice that the top attribute is "http". This is due to the large number of links in tweets. You can ignore links by using the filterlinks argument of CommentAttributes():

```sql
SELECT t.attribute, count(*) FROM (SELECT CommentAttributes(text USING PARAMETERS filterlinks=true) OVER(PARTITION BEST) FROM tweets) as t GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;
```

The attribute "http" is now gone from the list, but we still have "rt" (for retweet) on the list and it is not helpful in this context. You can omit terms such as "rt" by adding them to the stop_words list and reloading the stop_words user-dictionary:

```sql
INSERT INTO pulse.stop_words_en VALUES('rt');
commit;
SELECT LoadDictionary(standard USING PARAMETERS listName='stop_words') OVER() FROM pulse.stop_words_en;
```

When you rerun the query you get more accurate results for the popular topics in the data set:
SELECT t.attribute, count(*) FROM (SELECT CommentAttributes(text USING PARAMETERS filterlinks=true) OVER(PARTITION BEST) FROM tweets) as t GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;

<table>
<thead>
<tr>
<th>attribute</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>d11</td>
<td>4757</td>
</tr>
<tr>
<td>encryption</td>
<td>2356</td>
</tr>
<tr>
<td>perfume</td>
<td>2121</td>
</tr>
<tr>
<td>usb</td>
<td>1871</td>
</tr>
<tr>
<td>aes-256</td>
<td>1829</td>
</tr>
<tr>
<td>rt @hp</td>
<td>1788</td>
</tr>
<tr>
<td>world</td>
<td>1611</td>
</tr>
<tr>
<td>ceo</td>
<td>1542</td>
</tr>
<tr>
<td>interview</td>
<td>1346</td>
</tr>
<tr>
<td>cloud</td>
<td>1306</td>
</tr>
</tbody>
</table>

You can further refine the list to topics that contain specific attributes by adding the attributes in which you are interested to the white_list, and then filtering with the whitelist parameter:

SELECT t.attribute, count(*) FROM (SELECT CommentAttributes(text USING PARAMETERS filterlinks=true, whitelistonly=true) OVER(PARTITION BEST) FROM tweets) as t GROUP BY t.attribute ORDER BY count(*) DESC LIMIT 10;

Determining The Sentiment of Popular Topics

In addition to finding popular, or most discussed, topics in your data set, you can also easily get an average sentiment for the topics.

The following example uses a dataset of 10,000 tweets containing the hashtag #sports.

SELECT * from (SELECT attribute, count(attribute) AS cnt, AVG(sentiment_score) FROM (SELECT SentimentAnalysis(text USING PARAMETERS filterlinks=true) OVER(PARTITION BEST) from tweets) AS t1 GROUP BY t1.attribute ORDER BY AVG(sentiment_score) desc) AS t2 WHERE t2.cnt > 500 LIMIT 5;

The result shows the top 5 tweets with the highest average sentiment for attributes that have 500 or more occurrences:
<table>
<thead>
<tr>
<th>sport</th>
<th>count</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>football</td>
<td>817</td>
<td>0.290085679314565</td>
</tr>
<tr>
<td>game</td>
<td>638</td>
<td>0.134796238244514</td>
</tr>
<tr>
<td>baseball</td>
<td>1558</td>
<td>0.128369704749679</td>
</tr>
<tr>
<td>basketball</td>
<td>776</td>
<td>0.114690721649485</td>
</tr>
<tr>
<td>hockey</td>
<td>2610</td>
<td>0.113409961685824</td>
</tr>
</tbody>
</table>
Determining Prolific Authors

You can identify prolific authors of your textual data without using any of the Pulse functions. For example, using the same dataset as the examples in Determining Popular Topics, you can easily determine how many tweets were made by authors:

```sql
select "user.name", count(*) as post_count from tweets group by "user.name" order by count(*) DESC limit 10;
```

<table>
<thead>
<tr>
<th>user.name</th>
<th>post_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nick Cicero</td>
<td>182</td>
</tr>
<tr>
<td>Networked Society</td>
<td>171</td>
</tr>
<tr>
<td>AllThingsD</td>
<td>137</td>
</tr>
<tr>
<td>Stephanie~</td>
<td>117</td>
</tr>
<tr>
<td>Jennifer Ives</td>
<td>105</td>
</tr>
<tr>
<td>Claudia-ElasticMinds</td>
<td>101</td>
</tr>
<tr>
<td>Needful Things</td>
<td>96</td>
</tr>
<tr>
<td>Poptart Tech</td>
<td>85</td>
</tr>
<tr>
<td>Patrick Bertrand</td>
<td>84</td>
</tr>
<tr>
<td>Alessandro Piol</td>
<td>81</td>
</tr>
</tbody>
</table>

(10 rows)
Analyzing the Sentiment of Specific Authors

You can use the white_list feature of `SentimentAnalysis()` to filter the attributes so only the white_list terms are returned. You can combine the white_list with a query for a list of specific authors to narrow down the results to a specific subset of authors.

Using the same tweet_samples table in Analyzing Comments for a Company or Product, add the following sample tweets:

```
INSERT INTO tweets_sample VALUES('123', 'bcook', 'The hyperdrive is a great machine.');
INSERT INTO tweets_sample VALUES('124', 'sprock', 'The hyperdrive is a pinnacle of technology.');
INSERT INTO tweets_sample VALUES('125', 'tgates', 'what is a hyperdrive?');
INSERT INTO tweets_sample VALUES('126', 'bcook', 'Roses are red.');
INSERT INTO tweets_sample VALUES('127', 'sprock', 'Energy equals mass times the speed of light squared.');
INSERT INTO tweets_sample VALUES('128', 'tgates', 'Violets are blue.');
```

Create an authors table to hold the names of the authors whose sentiment you want to analyze:

```
CREATE TABLE authors (name VARCHAR, screenname VARCHAR);
```

Then insert the following authors:

```
INSERT INTO authors VALUES('Brian Cook', 'bcook');
INSERT INTO authors VALUES('Tom Gates', 'tgates');
INSERT INTO authors VALUES('Jim Sprock', 'sprock');
```

Add the word 'hyperdrive' to your existing white_list and reload the white_list user-dictionary:

```
INSERT INTO pulse.white_list_en VALUES('hyperdrive');
```

Then, you can run a query that filters on authors and the white_list and provides you with a sentiment score and the content of the analyzed text:

```
SELECT t1.id, t1.author, t1.attribute, t1.sentiment_score, t2.text FROM (SELECT id, author, SentimentAnalysis(text USING PARAMETERS whitelistonly=true) OVER (PARTITION BY id, author) FROM tweets_sample WHERE author IN (SELECT screenname FROM authors)) AS t1 JOIN (SELECT id, text FROM tweets_sample) AS t2 ON t1.id = t2.id;
```
<table>
<thead>
<tr>
<th>id</th>
<th>author</th>
<th>attribute</th>
<th>sentiment_score</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>bcook</td>
<td>hyperdrive</td>
<td>1</td>
<td>The hyperdrive is a great</td>
</tr>
<tr>
<td>124</td>
<td>sprock</td>
<td>hyperdrive</td>
<td>1</td>
<td>The hyperdrive is a pinnacle</td>
</tr>
<tr>
<td>125</td>
<td>tgates</td>
<td>hyperdrive</td>
<td>0</td>
<td>What is a hyperdrive?</td>
</tr>
</tbody>
</table>

(3 rows)
Finding Associated Attributes

Once you've analyzed your tweets and stored them in a table (see Batch Analyzing Data as It Is Loaded) you can use the analyzed data to make quick comparisons, such as finding attributes most associated with another attribute.

For example, if your primary attribute is 'microsoft', you may want to determine which other attributes are used most often with the word 'microsoft' in the same tweet. This can be accomplished with the following SQL:

```sql
select t1.attribute, count(*), avg(t1.sentiment_score) from tweet_sentiment t1, tweet_sentiment t2 where t1.id=t2.id and not t1.attribute=t2.attribute and t2.attribute = 'microsoft' group by t1.attribute order by count desc limit 5;
```

We get the following results from a data set of 25,000 PC Manufacturer tweets:

<table>
<thead>
<tr>
<th>attribute</th>
<th>count</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>windows phone</td>
<td>81</td>
<td>0.0238095238095238</td>
</tr>
<tr>
<td>power data center</td>
<td>77</td>
<td>0.58974358974359</td>
</tr>
<tr>
<td>wind project</td>
<td>77</td>
<td>0</td>
</tr>
<tr>
<td>investment</td>
<td>73</td>
<td>0</td>
</tr>
<tr>
<td>windows</td>
<td>57</td>
<td>0.175438596491228</td>
</tr>
</tbody>
</table>

The query allows you to gain additional insight into the scope of an attribute and may aid in determining the context of why a certain attribute it scored a certain way.
Using Pulse as an Aid in Competitive Analysis

This topic details how you can use Pulse to conduct basic competitive analysis for products or brands. Pulse makes basic competitive analysis simple through use of its white list feature. By utilizing the white list feature, you can analyze the tweets that pertain only to the brands or products that you are evaluating.

For example, say you wanted to analyze the sentiment of major food brands to determine how the brands compared to each other and what words people associate (positively and negatively) about the brands. Your work flow to do this analysis with Twitter and Vertica Pulse could be as follows:

1. Start collecting tweets based on the brands or products that you are following. For example, you can use the Social Media Connector (available on the Pulse marketplace) to collect tweets matching keywords.

2. First, create a white_list that contains the same keywords as the tweets that you are collecting. The whitelist allows you to later group and filter tweets collected. For example:

   ```sql
   insert into pulse.white_list_en values ('productA');
   insert into pulse.white_list_en values ('productB');
   insert into pulse.white_list_en values ('productC');
   \i /opt/vertica/packages/pulse/ddl/loadUserDictionaries.sql
   ```

3. **Batch Load Tweets**, and be sure to specify whitelistonly=true and relatedwords=true in the sentimentAnalysis() function. This creates a table with the sentiment score for your white-listed attributes. Note that this should be done in batches for large data sets. For smaller data sets (depending on your hardware) you can try and analyze all the tweets at once. For example:

   ```sql
   create table tweet_sentiment as
   (select id, "user.screen_name",
    SentimentAnalysis(text using parameters filterlinks=true, filterusermentions=true, relatedwords=true, filterretweets=true, whitelistonly=true)
    over (partition by id, "user.screen_name", text)
    from tweets where lang='en' order by attribute );
   ```
4. Verify that your tweet_sentiment table contains only your whitelist attributes. The following query should only return the brands/products that you have white listed. For example:

```sql
=> select distinct(attribute) from tweet_sentiment;

<table>
<thead>
<tr>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
</tr>
<tr>
<td>ProductB</td>
</tr>
<tr>
<td>ProductC</td>
</tr>
</tbody>
</table>
(3 rows)
```

5. You can get a basic idea of which product or brand is being talked about the most by seeing how many instances of each attribute appear in your data set:

```sql
=> select attribute, count(*) from tweet_sentiment group by (attribute) order by count(*) desc;

<table>
<thead>
<tr>
<th>attribute</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductA</td>
<td>701</td>
</tr>
<tr>
<td>ProductB</td>
<td>192</td>
</tr>
<tr>
<td>ProductC</td>
<td>52</td>
</tr>
</tbody>
</table>
(3 rows)
```

You can see that ProductA is the most talked about product of three being analyzed over the time-frame that the tweets were collected.

6. Determine the average sentiment scores of the tweets you have collected:

```sql
=> select attribute, avg(sentiment_score) as score from tweet_sentiment group by (attribute) order by score DESC;

<table>
<thead>
<tr>
<th>attribute</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductC</td>
<td>0.192307692307692</td>
</tr>
<tr>
<td>ProductB</td>
<td>-0.0729166666666667</td>
</tr>
<tr>
<td>ProductA</td>
<td>-0.122681883024251</td>
</tr>
</tbody>
</table>
(3 rows)
```

From this basic analysis, you can see that ProductC has the most positive sentiment from the three brands being analyzed over the time period when the tweets were collected, and ProductA has the lowest sentiment.

7. You can also determine which words or phrases are associated with each attribute in their positive and negative contexts. For example, to see the list of words that are most
associated with positive sentiment for ProductC, you can look at the related words fields and add up the occurrences of words associated with positive sentiment:

```sql
=> select count(*), related_word_1 from tweet_sentiment where attribute = 'ProductC' and sentiment_score > 0 group by related_word_1 order by count DESC;

<table>
<thead>
<tr>
<th>count</th>
<th>related_word_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>delicious</td>
</tr>
<tr>
<td>2</td>
<td>love</td>
</tr>
<tr>
<td>1</td>
<td>best</td>
</tr>
<tr>
<td>1</td>
<td>bless</td>
</tr>
<tr>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td>1</td>
<td>work</td>
</tr>
</tbody>
</table>

(6 rows)
```

You can also do the same for negative sentiment:

```sql
=> select count(*), related_word_1 from tweet_sentiment where attribute = 'ProductC' and sentiment_score < 0 group by related_word_1 order by count DESC;

<table>
<thead>
<tr>
<th>count</th>
<th>related_word_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>working</td>
</tr>
<tr>
<td>1</td>
<td>dragging</td>
</tr>
<tr>
<td>1</td>
<td>bad</td>
</tr>
<tr>
<td>1</td>
<td>doomed</td>
</tr>
<tr>
<td>1</td>
<td>loud</td>
</tr>
<tr>
<td>1</td>
<td>stressful</td>
</tr>
<tr>
<td>1</td>
<td>damn</td>
</tr>
</tbody>
</table>

(7 rows)
```

8. Finally, Pulse makes it easy to see other attributes associated with your target attributes to help you better understand the context in which people are discussing the brands or products that you are analyzing.

   a. Create another sentiment table from your data, but this time omit the `whitelistonly` and `relatedwords` parameters:

   ```sql
   create table tweet2_sentiment as
   (select id, "user.screen_name",
    SentimentAnalysis(text using parameters filterlinks=true,
    filterusermentions=true, filterretweets=true)
    over (partition by id, "user.screen_name", text)
    from tweets where lang='en' order by attribute );
   ```

   b. Next, query the tweets that contain your target attribute and find all the other attributes associated with those tweets. Display a count of the top 5 attributes (not
including the target attribute):

```sql
=> select count(attribute), attribute from tweet2_sentiment where id in (select id from tweet_sentiment where attribute = 'ProductC') and attribute <> 'ProductC' group by (attribute) order by count(attribute) DESC limit 5;
  count | attribute
--------
 13 | bbq
 11 | state
 11 | sandwich
 11 | steak
   3 | ProductB
(5 rows)
```

As you can see, a few basic queries can tell you the general sentiment differences between multiple brands or products. You can also determine which words are contributing to the sentiment of each product/brand that you are analyzing and which other attributes people are talking about when they mention the brand or product(s) that you are analyzing.

You could further refine these queries by breaking out different geographic locations or time of day by joining the IDs of the tweet_sentiment table back to the main tweets table and filtering be location or time.
Pulse Function Reference
**LoadDictionary**

Lists words from a Pulse user-defined dictionary into memory for use by sentimentAnalysis() and other Pulse functions.

This function must be used with the OVER() clause.

For more information on Pulse user-defined dictionaries, see [Dictionaries and Mappings](#).

**Syntax**

```sql
SELECT LoadDictionary(word USING PARAMETERS listName='listname'[, language='lang' [, label='label']])
OVER() FROM table;
```

**Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>A column of words to assign to a user-dictionary list. The column name must match the value of word.</td>
</tr>
<tr>
<td>listName</td>
<td>The user-dictionary list from which to load the values from <code>word</code>. Valid values:</td>
</tr>
<tr>
<td></td>
<td>• pos_words</td>
</tr>
<tr>
<td></td>
<td>• neg_words</td>
</tr>
<tr>
<td></td>
<td>• neutral_words</td>
</tr>
<tr>
<td></td>
<td>• stop_words</td>
</tr>
<tr>
<td></td>
<td>• white_list</td>
</tr>
<tr>
<td></td>
<td>See <a href="#">Dictionaries and Mappings</a> for details on each list type.</td>
</tr>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
</tbody>
</table>
### Argument Description

- **'spanish' or 'es'**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>The label that you want to assign to the dictionary.</td>
</tr>
<tr>
<td>table</td>
<td>The specified table from which values are loaded.</td>
</tr>
</tbody>
</table>

### Examples

```sql
SELECT LoadDictionary(standard USING PARAMETERS listName='neg_words_en') OVER() from pulse.neg_words_en;
SELECT LoadDictionary(standard USING PARAMETERS listName='pos_words_en') OVER() from pulse.pos_words_en;
SELECT LoadDictionary(standard USING PARAMETERS listName='pos_words_en', language='english') OVER() from pulse.pos_words_en;
SELECT LoadDictionary(standard USING PARAMETERS listName='pos_words_es', language='spanish') OVER() from pulse.pos_words_es;
SELECT LoadDictionary(standard USING PARAMETERS listName='neg_words',label='custom_negatives') OVER() from pulse.neg_words_en;
```

### See Also

- `LoadMapping()`
- `GetLoadedDictionary()`
- `GetStorage()`
LoadMapping

Loads a Pulse user-mapping into memory for use by sentimentAnalysis() and other Pulse functions.

This function must be used with the OVER() clause.

For more information on Pulse user-mappings, see Dictionaries and Mappings.

Syntax

```sql
SELECT LoadMapping(base, wordToMap USING PARAMETERS mapName='mapName' [, language='lang'][, label='label']) OVER() FROM table;
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>A column of base words to assign to a mapped word. The column name must match the value of base.</td>
</tr>
<tr>
<td>wordToMap</td>
<td>A column of words to map to the base word in the same row. The column name must match the value of wordToMap.</td>
</tr>
<tr>
<td>mapName</td>
<td>The mapping to load the words into.</td>
</tr>
<tr>
<td></td>
<td><strong>Valid values:</strong></td>
</tr>
<tr>
<td></td>
<td>- irregular_verbs — list of conjugations of verbs and their bases.</td>
</tr>
<tr>
<td></td>
<td>- normalization — list of synonyms and their base word.</td>
</tr>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td></td>
<td>- 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>- 'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>The label of the mapping that you want to load. If you do not provide a label, Pulse uses the default mapping.</td>
</tr>
<tr>
<td>table</td>
<td>The specified table from which values are loaded.</td>
</tr>
</tbody>
</table>
Examples

```sql
SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS mapName='normalization') OVER() from pulse.normalization_en;
SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS mapName='normalization', language='english') OVER() from pulse.normalization_en;
SELECT LoadMapping(standard_base, standard_synonym USING PARAMETERS mapName='normalization', language='spanish') OVER() from pulse.normalization_es;
```

See Also

- `LoadDictionary()`
- `GetLoadedMapping()`
- `GetStorage()`
**SentimentAnalysis**

Provides a sentiment score for each attribute (noun) in a given body of text. Positive sentiment receives a positive integer score and negative sentiment receives a negative integer score. A score of 0 indicates that the sentiment for the attribute is neutral.

This function must be used with the `OVER()` clause. Use `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `OVER()` clause. Any valid `PARTITION` `BY` clause is acceptable. However, only the `PARTITION` `BY` clause which matches the segmentation clause of the table's projection provides optimum performance. You can improve performance by segmenting on the columns in the `PARTITION` `BY` clause.

**Syntax**

```
SentimentAnalysis(text [, 'language'] [ USING PARAMETERS
[ whitelistonly = boolean ]
[, filterlinks = boolean ]
[, filterusermentions = boolean ]
[, filterhashtags = boolean ]
[, filterpunctuation = boolean ]
[, filterretweets = boolean ]
[, relatedwords = boolean ]
[, adjustcasing = boolean ]
[, language = string ]
[, label='Label']
[, granularity='ASD']
[, actionPattern='boolean']
])
```

**Note:** language can be specified as an argument and/or as a parameter. When specified as both, the argument value supersedes the parameter value.

**Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text to analyze. Limited to 65,000 bytes.</td>
</tr>
<tr>
<td>whitelistonly</td>
<td>Optional. Default false. When set to true only attributes defined in the whitelist user-dictionary are scored. Use this setting to limit your analysis to the objects of action patterns.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>filterlinks</td>
<td>Optional. Default false. When set to true, links are not included as attributes.</td>
</tr>
<tr>
<td>filterusermentions</td>
<td>Optional. Default false. When set to true, Twitter user mentions (@username) are not included as attributes.</td>
</tr>
<tr>
<td>filterhashtags</td>
<td>Optional. Default false. When set to true, Twitter hashtags (#hashtag) are not included as attributes.</td>
</tr>
<tr>
<td>filterpunctuation</td>
<td>Optional. Default true. Filters any punctuation that occurs at the beginning of an attribute other than @ and #.</td>
</tr>
<tr>
<td>filterretweets</td>
<td>Optional. Defaults to false. Filters out the characters &quot;RT&quot; from retweets in attributes.</td>
</tr>
<tr>
<td>relatedwords</td>
<td>Optional. Defaults to false. When set to true, provides up to three words from the sentence used to help determine the sentiment of the attribute.</td>
</tr>
<tr>
<td>adjustcasing</td>
<td>Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>• 'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>Optional. The label of the dictionaries that you want to use for sentiment analysis. If you do not include a label, Pulse uses the default dictionaries.</td>
</tr>
<tr>
<td>granularity</td>
<td>Optional. The level of the sentiment analysis that you want to perform:</td>
</tr>
<tr>
<td></td>
<td>• A — Attribute level analysis</td>
</tr>
<tr>
<td></td>
<td>• S — Sentence level analysis</td>
</tr>
</tbody>
</table>
Argument | Description
--- | ---
| D — Document level analysis
You can specify any granularity level or combination of levels with your sentiment analysis. If you do not specify a granularity level, Pulse performs an attribute level analysis.

ActionPattern | Optional. Default false. When set to true checks for action patterns in the analyzed content.

Examples

These examples show various ways you can use Pulse to detect user sentiment.

Query for sentiment in the following sentence.

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
<td>-1</td>
</tr>
</tbody>
</table>

(2 rows)

Query to identify the words that triggered the sentiment score.

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.' USING PARAMETERS relatedwords=true) OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
<th>related_word_1</th>
<th>related_word_2</th>
<th>related_word_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
<td>1</td>
<td>quick</td>
<td>lazy</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
<td>-1</td>
<td>lazy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.', 'english') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
<td>-1</td>
</tr>
</tbody>
</table>

(2 rows)

```
SELECT SentimentAnalysis('The quick brown fox jumped over the lazy dog.' using PARAMETERS language='english') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
<td>-1</td>
</tr>
</tbody>
</table>
(2 rows)

```sql
SELECT SentimentAnalysis('El zorro rapido brinco sobre el perro flojo.', 'spanish') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zorro</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
<td>-1</td>
</tr>
</tbody>
</table>

(2 rows)

```sql
SELECT SentimentAnalysis('El zorro rapido brinco sobre el perro flojo.' using PARAMETERS language='spanish') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zorro</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
<td>-1</td>
</tr>
</tbody>
</table>

(2 rows)

```sql
SELECT SentimentAnalysis('The camera takes great quality pictures but is expensive. It feels like a professional one.' USING PARAMETERS granularity='ASD') over();
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
<th>mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>camera</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>quality pictures</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

SELECT sentimentAnalysis('Right after school on November 8th I will go to target, walmart, and best buy and buy #blueslidepark just for @MacMiller' USING PARAMETERS actionPattern=true,whitelistonly=true) over();

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment_score</th>
<th>action</th>
<th>action_pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>walmart</td>
<td>1</td>
<td>go to target</td>
<td>#action{$verb $prep $verb}</td>
</tr>
<tr>
<td>1</td>
<td>walmart</td>
<td>1</td>
<td>go to target</td>
<td>#action{$verb to $verb}</td>
</tr>
</tbody>
</table>

(2 rows)

**Getting Twitter User-Mentioned Sentiment**

```sql
SELECT SentimentAnalysis('@company is great!') OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
<th>sentiment score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>@company</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)

**Filtering Twitter User Sentiment**
See Also

- LoadDictionary()
- LoadMapping()
- ExtractSentence()
- GetSentenceCount()
- GetAllSentences()
- CommentAttributes()
PartsOfSpeech

Tags the words in one or more sentences with their part of speech classification, using Penn Treebank parts of speech tags.

Syntax

```
SELECT PartsOfSpeech('sentences', language='lang') [using PARAMETERS [language='lang'],
adjustcasing=boolean] OVER(PARTITION BEST);
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentences</td>
<td>One or more sentences to be tagged with parts of speech markup.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>'spanish' or 'es'</td>
</tr>
<tr>
<td>adjustcasing</td>
<td>Optional. Defaults to false. When set to true, all letters in the text are</td>
</tr>
<tr>
<td></td>
<td>converted to uppercase before sentence detection. After performing sentence</td>
</tr>
<tr>
<td></td>
<td>detection, Vertica converts all letter to lowercase. This option can help</td>
</tr>
<tr>
<td></td>
<td>you in cases where the original data is all in lowercase letters and Pulse</td>
</tr>
<tr>
<td></td>
<td>is incorrectly identifying sentence boundaries.</td>
</tr>
</tbody>
</table>

Notes

- This function returns a part of speech markup for each word. The markup used is the Penn Treebank Project Parts of Speech Tags while for Spanish the Parole Reduced Tagset is used.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
## Examples

```sql
select partsOfSpeech('The quick brown fox jumped over the lazy dog.') OVER(PARTITION BEST);
```
```
sentence | token | part_of_speech
---------|-------|-----------------|
1 | the | DT
1 | quick | JJ
1 | brown | JJ
1 | fox | NN
1 | jumped | VBD
1 | over | IN
1 | the | DT
1 | lazy | JJ
1 | dog | NN
1 | . | .
```

(10 rows)

```sql
select partsOfSpeech('Every good boy deserves fudge.') OVER(PARTITION BEST);
```
```
sentence | token | part_of_speech
---------|-------|-----------------|
1 | every | DT
1 | good | JJ
1 | boy | NN
1 | deserves | VBD
1 | fudge | NN
1 | . | .
```

(6 rows)

```sql
select partsOfSpeech('The quick brown fox jumped over the lazy dog.', 'english') OVER(PARTITION BEST);
```
```
sentence | token | part_of_speech
---------|-------|-----------------|
1 | the | DT
1 | quick | JJ
1 | brown | JJ
1 | fox | NN
1 | jumped | VBD
1 | over | IN
1 | the | DT
1 | lazy | JJ
1 | dog | NN
1 | . | .
```

(10 rows)

```sql
select partsofSpeech('El zorro rapido brinco sobre el perro flojo','spanish') OVER();
```
```
sentence | token | part_of_speech
---------|-------|-----------------|
1 | El | DA
1 | zorro | NC
1 | rapido | AQ
1 | brinco | AQ
```

<table>
<thead>
<tr>
<th></th>
<th>sobre</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>el</td>
<td>DA</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
<td>NC</td>
</tr>
<tr>
<td>1</td>
<td>flojo</td>
<td>AQ</td>
</tr>
</tbody>
</table>

(8 rows)

See Also

- SentimentAnalysis()
GetAllDictionarySetLabels

Lists all the dictionary labels that are loaded into the current Pulse session. This function shows you which labels are currently in use. You can load only one dictionary of each type in a single session.

Syntax

SELECT GetAllDictionarySetLabels() OVER();

Examples

```
SELECT GetAllDictionarySetLabels() OVER();
  label
  --------
  default
  sports_teams
(2 rows)
```
GetAllDictionaryWords

Lists all dictionary words that are currently loaded into Pulse. This function can help you determine which user-defined words in a sentence might be affecting the sentiment score of an attribute.

Syntax

```
SELECT GetAllDictionaryWords([using PARAMETERS language='language'[, label='Label']]) OVER();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td></td>
<td>'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>The label of the dictionaries that you want to list. If you do not provide a label, Pulse uses the default dictionaries.</td>
</tr>
</tbody>
</table>

Examples

```
SELECT GetAllDictionaryWords() OVER();
```

<table>
<thead>
<tr>
<th>dictionary</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
select GetAllDictionaryWords(using parameters language='english') over();
```

<table>
<thead>
<tr>
<th>dictionary</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
select GetAllDictionaryWords(using parameters label='music') over();
```

<table>
<thead>
<tr>
<th>dictionary</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>white_list_en</td>
<td>classical</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>white_list_en</td>
<td>popular</td>
</tr>
<tr>
<td>white_list_en</td>
<td>rock</td>
</tr>
</tbody>
</table>

(3 rows)

See Also

- [GetAllMappingWords()](#)
GetAllLoadedDictionaries

Lists all the dictionaries and dictionary labels that are loaded into the current Pulse session. This function shows you which dictionaries are determining the sentiment score of an attribute. Only one dictionary of each type can be loaded in a single session.

Syntax

```
SELECT GetAllLoadedDictionaries() OVER();
```

Examples

```
SELECT GetAllLoadedDictionaries() OVER();

<table>
<thead>
<tr>
<th>dictionary</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>neg_words_en</td>
<td>default</td>
</tr>
<tr>
<td>stop_words_es</td>
<td>default</td>
</tr>
<tr>
<td>neutral_words_es</td>
<td>default</td>
</tr>
<tr>
<td>white_list_en</td>
<td>default</td>
</tr>
<tr>
<td>normalization_en</td>
<td>default</td>
</tr>
<tr>
<td>pos_words_es</td>
<td>default</td>
</tr>
<tr>
<td>neg_words_es</td>
<td>default</td>
</tr>
<tr>
<td>pos_words_en</td>
<td>default</td>
</tr>
<tr>
<td>white_list_es</td>
<td>default</td>
</tr>
<tr>
<td>neutral_words_en</td>
<td>default</td>
</tr>
<tr>
<td>stop_words_en</td>
<td>default</td>
</tr>
<tr>
<td>normalization_es</td>
<td>default</td>
</tr>
</tbody>
</table>

(12 rows)
```
GetAllMappingWords

Lists all user-defined bases and synonyms that are currently loaded into Pulse. This function helps you determine which user-defined mappings in a sentence might be affecting the sentiment score of an attribute.

Syntax

```
SELECT GetAllMappingWords([using PARAMETERS language='language'][, label='label']) OVER();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td></td>
<td><em>'english' or 'en'</em></td>
</tr>
<tr>
<td></td>
<td><em>'spanish' or 'es'</em></td>
</tr>
<tr>
<td>label</td>
<td>The label of the mappings that you want to list. If you do not provide a label, Pulse uses the default dictionaries.</td>
</tr>
</tbody>
</table>

Examples

```
SELECT GetAllMappingWords() OVER() limit 10;
```

<table>
<thead>
<tr>
<th>mapping</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalization</td>
<td>hp</td>
<td>hewlett packard</td>
</tr>
<tr>
<td>normalization</td>
<td>hp</td>
<td>hewlett-packard</td>
</tr>
<tr>
<td>normalization</td>
<td>companycorp</td>
<td>company-corp</td>
</tr>
<tr>
<td>normalization</td>
<td>companycorp</td>
<td>companycorps</td>
</tr>
<tr>
<td>normalization</td>
<td>companycorp</td>
<td>companycorp's</td>
</tr>
<tr>
<td>normalization</td>
<td>producthd</td>
<td>product hd</td>
</tr>
<tr>
<td>normalization</td>
<td>producthd</td>
<td>product-hd</td>
</tr>
<tr>
<td>normalization</td>
<td>companycorp</td>
<td>company corp</td>
</tr>
</tbody>
</table>

(8 rows)
See Also

- GetAllDictionaryWords()
CommentAttributes

Retrieves the attributes (nouns) from a given piece of text.

Syntax

CommentAttributes(text[, language][ USING PARAMETERS
    [ whitelistonly = boolean ]
    [, filterlinks = boolean ]
    [, filterusermentions = boolean ]
    [, filterhashtags = boolean ]
    [, filterpunctuation = boolean ]
    [, filterretweets = boolean ]
    [, adjustcasing = boolean ]
    [, language = string ]])
]

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text from which to extract the attributes.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>• 'spanish' or 'es'</td>
</tr>
<tr>
<td>whitelistonly</td>
<td>Optional. Default false. When set to true only attributes defined in the</td>
</tr>
<tr>
<td></td>
<td>white_list user-dictionary are returned.</td>
</tr>
<tr>
<td>filterlinks</td>
<td>Optional. Default false. When set to true, links are not set as attributes.</td>
</tr>
<tr>
<td>filterusermentions</td>
<td>Optional. Default false. When set to true, Twitter usernames (@username) are not set as attributes.</td>
</tr>
<tr>
<td>filterhashtags</td>
<td>Optional. Default false. When set to true, removes the following from tweets:</td>
</tr>
<tr>
<td></td>
<td>• hashtag symbols - For example, #pizza becomes pizza.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| @mentions           | • For example, Vertica would remove @NewYorkCity from a tweet.  
|                     | • Link URLs                                                                                                                                   |
| filterpunctuation   | Optional. Default true. Filters any punctuation that occurs at the beginning of an attribute other than @ and #. |
| filterretweets      | Optional. Defaults to false. Filters out the characters "RT" from retweets in attributes.                                                  |
| adjustcasing        | Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries. |

### Notes

- The text argument is limited to 65,000 bytes.

- This function must be used with the over() clause. Use with OVER(PARTITION BEST) for the best performance if the query does not require specific columns in the over() clause.

- *language* can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

### Examples

```sql
SELECT CommentAttributes('The quick brown fox jumped over the lazy dog. All good boys deserve fudge. ') OVER(PARTITION BEST);
sentence | attribute  
----------|------------
          | fox        
          | dog        
          | boys       
          | fudge      
(4 rows)  
```
SELECT commentattributes('the quick brown fox jumped over the lazy dog. All good boys deserve fudge', 'english') OVER();

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
</tr>
<tr>
<td>2</td>
<td>boys</td>
</tr>
<tr>
<td>2</td>
<td>fudge</td>
</tr>
</tbody>
</table>

(4 rows)

SELECT commentattributes('the quick brown fox jumped over the lazy dog. All good boys deserve fudge' using parameters language='english') OVER();

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fox</td>
</tr>
<tr>
<td>1</td>
<td>dog</td>
</tr>
<tr>
<td>2</td>
<td>boys</td>
</tr>
<tr>
<td>2</td>
<td>fudge</td>
</tr>
</tbody>
</table>

SELECT commentattributes('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio', 'spanish') OVER();

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zorro</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
</tr>
<tr>
<td>2</td>
<td>chicos</td>
</tr>
<tr>
<td>2</td>
<td>premio</td>
</tr>
</tbody>
</table>

(4 rows)

SELECT commentattributes('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio' using PARAMETERS language='spanish') OVER();

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zorro</td>
</tr>
<tr>
<td>1</td>
<td>perro</td>
</tr>
<tr>
<td>2</td>
<td>chicos</td>
</tr>
<tr>
<td>2</td>
<td>premio</td>
</tr>
</tbody>
</table>

(4 rows)

Filtering User-mentions

SELECT CommentAttributes('@user is always late. He kept me waiting 20 minutes last weekend.' USING PARAMETERS filterusermentions=true) OVER(PARTITION BEST);

<table>
<thead>
<tr>
<th>sentence</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>weekend</td>
</tr>
</tbody>
</table>

(1 row)

See Also

- SentimentAnalysis()
**GetSentenceCount**

Returns the number of sentences in a body of text. You can use this function to count the number of sentences in a long piece of text. It is also useful if you are programatically using the `ExtractSentence` function and need to know the number of sentences in a piece of text.

**Syntax**

```
select GetSentenceCount(text [, language] [ USING PARAMETERS
[ filterlinks = boolean ]
[, filterusermentions = boolean ]
[, filterhashtags = boolean ]
[, adjustcasing = boolean ]
[, language = string ]
])
```

**Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text from which to extract the number of sentences. Currently English and Spanish language text are supported for analysis.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>* 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>* 'spanish' or 'es'</td>
</tr>
<tr>
<td>filterlinks</td>
<td>Optional. Default false. When set to true, sentences that are only links are not counted as a sentence.</td>
</tr>
<tr>
<td>filterusermentions</td>
<td>Optional. Default false. When set to true, sentences that are only Twitter user mentions (@username) are not counted as a sentence.</td>
</tr>
<tr>
<td>filterhashtags</td>
<td>Optional. Default false. When set to true, sentences that are only Twitter hashtags (#hashtag) are not counted as a sentence.</td>
</tr>
<tr>
<td>adjustcasing</td>
<td>Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.</td>
</tr>
</tbody>
</table>

**Notes**

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- `language` can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

**Examples**

```sql
SELECT GetSentenceCount('The quick brown fox jumped over the lazy dog. Every good boy deserves fudge') OVER(PARTITION BEST);
sentence_count
-----------------
  2
(1 row)

SELECT getsentencecount('http://hp.com. @hp. http://hp.com is great!') OVER(PARTITION BEST);
sentence_count
-----------------
  3
(1 row)

SELECT getsentencecount('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio' using PARAMETERS language='spanish') over();
sentence_count
-----------------
  2
(1 row)

SELECT getsentencecount('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio','spanish') over();
sentence_count
-----------------
  2
(1 row)
```
Filtering Links and User Mentions

```
SELECT GetSentenceCount('http://hp.com. @hp. http://hp.com is great!' USING PARAMETERS filterlinks=true, filterusermentions=true) OVER(PARTITION BEST);
```

```
sentence_count
------------------
        1
(1 row)
```

See Also

- `GetAllSentences()`
- `ExtractSentence()`
**ExtractSentence**

Returns the specified sentence from a body of text.

**Syntax**

```
ExtractSentence(text, sentence [, Language] [USING PARAMETERS
[ filterlinks = boolean ]
[, filterusermentions = boolean ]
[, filterhashtags = boolean ]
[, adjustcasing = boolean ]
[, language = string ]
])
```

**Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text containing the sentence to extract.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>* 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>* 'spanish' or 'es'</td>
</tr>
<tr>
<td>sentence</td>
<td>Integer value. The number of the sentence in the text.</td>
</tr>
<tr>
<td>filterlinks</td>
<td>Optional. Default false. When set to true, sentences that are only links are skipped over and ignored. Any links in a sentence are not included in the extracted sentence.</td>
</tr>
<tr>
<td>filterusermentions</td>
<td>Optional. Default false. When set to true, sentences that are only Twitter user mentions (@username) are skipped over and ignored. Any user-mentions in a sentence are not included in the extracted sentence.</td>
</tr>
<tr>
<td>filterhashtags</td>
<td>Optional. Default false. When set to true, sentences that are only Twitter hashtags (#hashtag) are skipped over and ignored. Any hashtags in a sentence are not included in the extracted sentence.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>adjustcasing</td>
<td>Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.</td>
</tr>
</tbody>
</table>

### Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the `over()` clause. Use with `OVER(PARTITION BEST)` for the best performance if the query does not require specific columns in the `over()` clause.
- `language` can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

### Examples

```sql
SELECT ExtractSentence('The quick brown fox jumped. Every good boy deserves fudge', 2) OVER(PARTITION BEST);
sentence
------------------------------------------
Every good boy deserves fudge.
(1 row)
```

```sql
SELECT ExtractSentence('the quick brown fox jumped over the lazy dog. All good boys deserve fudge', 2, 'english') OVER();
sentence
------------------------------------------
All good boys deserve fudge
(1 row)
```

```sql
SELECT ExtractSentence('the quick brown fox jumped over the lazy dog. All good boys deserve fudge', 2, 'spanish') OVER();
sentence
------------------------------------------
All good boys deserve fudge
(1 row)
```

```sql
SELECT ExtractSentence('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio', 2, 'spanish') OVER();
sentence
------------------------------------------
```

Todos los chicos buenos merecen un premio
(1 row)

SELECT extractSentence('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio', 2, using parameters language='spanish') OVER();

Todos los chicos buenos merecen un premio
(1 row)

Filtering Links

SELECT ExtractSentence('HP - http://hp.com is a useful website. I like HP.', 1, USING PARAMETERS filterlinks=true) OVER(PARTITION BEST);

hp - is a useful website.
(1 row)

See Also

- GetSentenceCount()
- GetAllSentences()
GetAllSentences

Extracts a row for each sentence in a body of text. This ability is useful if you need to programmatically get each sentence in a piece of text.

Syntax

GetAllSentences(text [, Language[ USING PARAMETERS 
[ filterlinks = boolean ] 
[, filterusermentions = boolean ] 
[, filterhashtags = boolean ] 
[, adjustcasing = boolean ] 
[, language = string ] ])

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text from which to get the sentences.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>- 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>- 'spanish' or 'es'</td>
</tr>
<tr>
<td>filterlinks</td>
<td>Optional. Default false. When set to true, sentences that are only links</td>
</tr>
<tr>
<td></td>
<td>are skipped over and ignored. Any links in a sentence are not included in</td>
</tr>
<tr>
<td>filterusermentions</td>
<td>Optional. Default false. When set to true, sentences that are only</td>
</tr>
<tr>
<td></td>
<td>Twitter user mentions (@username) are skipped over and ignored. Any user-</td>
</tr>
<tr>
<td></td>
<td>mentions in a sentence are not included in the extracted sentence.</td>
</tr>
<tr>
<td>filterhashtags</td>
<td>Optional. Default false. When set to true, sentences that are only</td>
</tr>
<tr>
<td></td>
<td>Twitter hashtags (#hashtag) are skipped over and ignored. Any hashtags in</td>
</tr>
<tr>
<td></td>
<td>a sentence are not included in the extracted sentence.</td>
</tr>
</tbody>
</table>
## Argument Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustcasing</td>
<td>Optional. Defaults to false. When set to true, all letters in the text are converted to uppercase before sentence detection. After performing sentence detection, Vertica converts all letter to lowercase. This option can help you in cases where the original data is all in lowercase letters and Pulse is incorrectly identifying sentence boundaries.</td>
</tr>
</tbody>
</table>

### Notes

- The text argument is limited to 65,000 bytes.
- This function must be used with the over() clause. Use with OVER(PARTITION BEST) for the best performance if the query does not require specific columns in the over() clause.
- `language` can be specified as an argument and/or as a parameter where the argument value supersedes the parameter value.

### Examples

```sql
SELECT GetAllSentences('The quick brown fox jumped over the lazy dog. Every good boy deserves fudge') OVER(PARTITION BEST);
```

```
sentence
-------------------
The quick brown fox jumped over the lazy dog.
Every good boy deserves fudge.
(2 rows)
```

```sql
SELECT getAllSentences('the quick brown fox jumped over the lazy dog. All good boys deserve fudge','english') OVER();
```

```
sentence_index | sentence_text
-------------|----------------
1             | the quick brown fox jumped over the lazy dog.
2             | All good boys deserve fudge
(2 rows)
```

```sql
SELECT getAllSentences('the quick brown fox jumped over the lazy dog. All good boys deserve fudge' using parameters language='english') OVER ();
```

```
sentence_index | sentence_text
-------------|----------------
1             | the quick brown fox jumped over the lazy dog.
2             | All good boys deserve fudge
```
(2 rows)

```sql
SELECT GetAllSentences('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio' , 'spanish') OVER();
```

<table>
<thead>
<tr>
<th>sentence_index</th>
<th>sentence_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>el zorro rapido brinco sobre el perro flojo.</td>
</tr>
<tr>
<td>2</td>
<td>Todos los chicos buenos merecen un premio</td>
</tr>
</tbody>
</table>

(2 rows)

```sql
SELECT GetAllSentences('el zorro rapido brinco sobre el perro flojo. Todos los chicos buenos merecen un premio' USING parameters language='spanish') over();
```

<table>
<thead>
<tr>
<th>sentence_index</th>
<th>sentence_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>el zorro rapido brinco sobre el perro flojo.</td>
</tr>
<tr>
<td>2</td>
<td>Todos los chicos buenos merecen un premio</td>
</tr>
</tbody>
</table>

(2 rows)

### Filtering User-mentions

```sql
SELECT GetAllSentences('@user is always late. He kept me waiting 20 minutes last time.' USING PARAMETERS filterusermentions=true) OVER(PARTITION BEST);
```

<table>
<thead>
<tr>
<th>sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>is always late.</td>
</tr>
<tr>
<td>he kept me waiting 20 minutes last time.</td>
</tr>
</tbody>
</table>

(2 rows)

### See Also

- [GetSentenceCount()](#)
- [ExtractSentence()](#)
SetDefaultLanguage

Sets the new default language to use for Pulse functions if no language is specified in a Pulse function call.

Syntax

SetDefaultLanguage(Language)

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>• 'spanish' or 'es'</td>
</tr>
</tbody>
</table>

Notes

- This function must be used with the OVER() clause.
- The default language immediately after installation is English.
- The language that is set when using this function is the default language across all sessions and is persistent across database restarts.

Examples

=> select setDefaultLanguage('es') over();
Success
--------
t
(1 row)
See Also

- SentimentAnalysis
GetLoadedDictionary

Lists the currently loaded words for the specified user-dictionary.

If the user-dictionary is not loaded, then nothing is returned. You must use the OVER() clause with this function.

Syntax

```
SELECT GetLoadedDictionary(user-dictionary [using PARAMETERS language = string][, label='label']) OVER ();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-dictionary</td>
<td>The user-dictionary list to retrieve.</td>
</tr>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td>label</td>
<td>The label of the dictionaries that you want to list. If you do not provide a label, Pulse uses the default dictionaries.</td>
</tr>
</tbody>
</table>

**Valid values:**

- `pos_words`
- `neg_words`
- `neutral_words`
- `stop_words`
- `white_list`

See [Dictionaries and Mappings](#) for details on each type.
Examples

Note: This example is from a three node cluster, so three copies of the words are returned.

```sql
SELECT GetLoadedDictionary('pos_words') OVER();
------
word
---
 :-) ade\t赞赏
 admire    \t羡慕
 admiringly    \t羡慕
 adore     \t喜爱
 adoringly    \t喜爱
 adulation \t爱恋
 adventuresome \t热爱
 advocated \t崇拜
 affable \t友好
 affably \t友好
 affordable \t可负担
 affordably \t可负担
 affordable \t可负担
 all-around \t广泛
 alluringly \t迷人
 amazement \t惊叹
 ameliorate \t改良
 ample \t充裕
 amusing \t令人愉快
--More--
```

```sql
SELECT GetLoadedDictionary('pos_words' using PARAMETERS language='english') OVER();
------
word
---
simplicity
(1 row)
```

```sql
SELECT GetLoadedDictionary('pos_words' using PARAMETERS language='spanish') OVER();
------
simplidad
(1 row)
```

See Also

- `LoadDictionary()`
- `GetLoadedMapping()`
GetLoadedMapping

Lists the currently loaded words for the specified user-defined mapping.

If the mapping is not loaded with LoadMapping, then nothing is returned. This function must be used with the OVER() clause.

Syntax

```
SELECT GetLoadedMapping('normalization' [using PARAMETERS language = string][, label='label']) OVER();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapping</td>
<td>The mapping list to retrieve. Currently the only mapping supported is: normalization</td>
</tr>
<tr>
<td></td>
<td><strong>Note: By default, the normalization list is empty.</strong></td>
</tr>
<tr>
<td>language</td>
<td>The language of the dictionary:</td>
</tr>
<tr>
<td></td>
<td>- 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>- 'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>The label to which you want to load the specified mapping. If you do not include a label, Pulse loads the default UDDs.</td>
</tr>
</tbody>
</table>

Examples

```
SELECT GetLoadedMapping('normalization') OVER();
```

```
key    | value
-------|-------
hp     | hewlett packard
(1 row)
```
### See Also

- `LoadMapping()`
- `GetLoadedDictionary()`
GetStorage

Lists the currently loaded user-dictionaries and user-defined mapping. This function must be used with the OVER() clause.

Syntax

SELECT GetStorage([using PARAMETERS label='label']) OVER();

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>The label of the dictionaries and mapping names that you want to list. If you do not provide a label, Pulse uses the default dictionaries.</td>
</tr>
</tbody>
</table>

Examples

```
SELECT GetStorage() OVER();

key
---
key
neg_words_en
neutral_words_en
pos_words_en
stop_words_en
white_list_en
normalization_en
neg_words_es
neutral_words_es
pos_words_es
stop_words_es
white_list_es
normalization_es
(12 rows)
```
See Also

- `LoadDictionary()`
- `LoadMapping()`
- `GetLoadedDictionary()`
- `GetLoadedMapping()`
UnloadLabeledDictionary

Unloads a specific dictionary from a Pulse session. The dictionary continues to exist, and a user can later reload the dictionary, if needed.

You cannot unload a default dictionary, but you can replace it by loading a custom user-defined dictionary.

Syntax

```
SELECT unloadLabeledDictionary(USING PARAMETERS listname='listname', language='lang', label='label') OVER();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listName</td>
<td>The type of the dictionary that you want to unload. listName must be one of:</td>
</tr>
<tr>
<td></td>
<td>• pos_words</td>
</tr>
<tr>
<td></td>
<td>• neg_words</td>
</tr>
<tr>
<td></td>
<td>• neutral_words</td>
</tr>
<tr>
<td></td>
<td>• stop_words</td>
</tr>
<tr>
<td></td>
<td>• white_list</td>
</tr>
<tr>
<td></td>
<td>See Dictionaries and Mappings for details on each list type.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>• 'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>The label of the dictionary that you want to unload.</td>
</tr>
</tbody>
</table>
Examples

```sql
select unloadLabeledDictionary(USING PARAMETERS listname='neg_words', label='custom_negatives') OVER();

success
---------
t
(1 row)
```

See Also

- `UnloadLabeledDictionarySet()`
UnloadLabeledDictionarySet

Unloads all user-defined dictionaries with a particular label from a Pulse session. The dictionaries continue to exist, and a user can later reload the dictionaries, if needed.

You cannot unload a default dictionary, but you can replace it by loading a custom user-defined dictionary.

Syntax

```sql
SELECT unloadLabeledDictionarySet(USING PARAMETERS label='labelName') OVER();
```

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>label</code></td>
<td>The label of the dictionary set that you want to unload.</td>
</tr>
</tbody>
</table>

Examples

```sql
select unloadLabeledDictionarySet(USING PARAMETERS label='custom_negatives') OVER();
```

```
success
-------
t
(1 row)
```

See Also

- `UnloadLabeledDictionary()`
UnloadLabeledMapping

Unloads a specific mapping from a Pulse session. The mapping continues to exist, and a user can later reload it, if needed.

Syntax

SELECT unloadLabeledMapping(USING PARAMETERS mapName='normalization' [, language='Lang'][, label='label']) OVER();

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapName</td>
<td>The name of the mapping from which you are unloading the dictionary.</td>
</tr>
<tr>
<td>language</td>
<td>The language:</td>
</tr>
<tr>
<td></td>
<td>• 'english' or 'en'</td>
</tr>
<tr>
<td></td>
<td>• 'spanish' or 'es'</td>
</tr>
<tr>
<td>label</td>
<td>The label of the mapping that you want to unload.</td>
</tr>
</tbody>
</table>

Examples

```
select unloadLabeledMapping(standard USING PARAMETERS label='custom_mapping') OVER();

success
--------
t
(1 row)
```
Welcome to the Vertica Plug-in for Informatica Guide.
Introduction to Using Informatica PowerCenter with Vertica

Informatica's PowerCenter family of products lets you collect, transform, and store data. The products support a wide variety of data sources, including databases, message queues, and many different file formats.

The PowerCenter Client consists of four main applications:

- Use Designer to create sources, targets, and mappings.
- Use Workflow Manager to create workflows for those sources, targets and mapping you created in Designer.
- Use Workflow Monitor to monitor running workflows.
- Use Repository Manager to manage repository resources, such as moving folders and objects and managing permissions and users.

The PowerCenter Server enables you to access, read, and write to Vertica.

PowerCenter 9.6.1 HotFix 2 includes the PowerExchange (PWX) Connector for Vertica. Informatica developed this PWX Connector as an alternative to the Vertica plug-in. The PWX connection includes additional capabilities and performance improvements when connected to Vertica.

History of Integration Between Vertica and Informatica

Prior to the PowerExchange (PWX) Connector for Vertica, Vertica developed and supported two connectors for Informatica and Vertica:

- In 2009, Vertica developed the Vertica Plug-in for Informatica. This connector used the native method of loading data from Informatica into Vertica.

- In 2013, Vertica replaced the earlier Vertica Plug-in for Informatica with a Java plug-in that supports all operating system platforms. This plug-in runs on generic JDBC and ODBC connections and includes new and improved features compared to the native plug-in.
This document describes how to use the Vertica Java plug-in for connecting from Informatica to Vertica.

In 2014, Informatica released the PWX Connector for Vertica. This connector includes enhancements to the partitioning and pushdown capabilities of Informatica.

The following timeline shows the history of plug-ins for connecting from Informatica to Vertica:

Vertica recommends that you use the new PWX Connector for Vertica with Informatica PowerCenter 9.6.1 HotFix 2 or later to connect to your Vertica database.

For detailed information about using the PWX Connector for Vertica, see Vertica Integration with Informatica: Connection Guide.

How the Plug-in Is Configured with Vertica and Informatica PowerCenter

The following illustration provides an overview of the configuration.
This manual provides information for installing plug-in components, using the plug-in to access Vertica as source or target database, and implementing and modifying plug-in features.

**Overview**

The following table provides recommended steps.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Follow the procedures in <a href="#">Installing the Vertica Plug-in for PowerCenter</a> to download and install plug-in</td>
<td>You must install both client and server components.</td>
</tr>
<tr>
<td>Step</td>
<td>Action</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>components.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Review the sample for using the plug-in with PowerCenter in <em>Using the Vertica Plug-in with Informatica PowerCenter</em>.</td>
<td>The sample shows how to import Vertica as source and target. I also provides steps that show you how to specify and connect to your Vertica database and include it in your workflows.</td>
</tr>
<tr>
<td>3</td>
<td>Set plug-in features according to your specific needs. Features are listed and described in <em>Accessing and Setting Plug-in Features</em>.</td>
<td>Be aware of situations where the use of one feature depends on another. For example, to take advantage of the increased performance of <em>EnableStreamingBatchInsert</em>, you must set <em>Copy Local Method</em> to <em>None</em>.</td>
</tr>
<tr>
<td>4</td>
<td>Check <em>Best Practices</em> for required memory settings and other tips.</td>
<td>Memory requirements are highly dependent upon Informatica and Vertica dedicated resources.</td>
</tr>
</tbody>
</table>
Installing the Vertica Plug-in for PowerCenter

You must download and install a client and a server component for the Vertica Plug-in for Informatica.

As a first step, download both the client and server components of the plug-in from the myVertica portal.

The client portion of the plug-in (vertica-informatica-client-2.0.1.zip) includes the following files:

- `vertica.xml` contains the metadata definition needed by the PowerCenter repository to allow communication between PowerCenter and Vertica.

- `verticacli9.dll` is a library file you add to your Windows registry.

- `vertica-infa951-x86.reg` is a registry file you can use to register the dll on 32-bit machines for Informatica PowerCenter 9.5.1.

- `vertica-infa951-x64.reg` is a registry file you can use to register the dll on 64-bit machines for Informatica PowerCenter 9.5.1.

The server portion includes the file `vertica-informatica-plugin-server.jar`.

Note: Each server type requires the Java 6.0 run-time environment.

Installing the Vertica plug-in is a multi-step process. The following sections explain these steps in greater detail, using a simple example.

1. Register the plug-in's metadata with the PowerCenter Repository Service with which you want to access Vertica. Follow the procedures in Registering the Plug-in's Metadata.

2. Add the `verticacli9.dll` library file to the Windows registry. Follow the procedures in Adding the Library File to the Windows Registry.

3. Copy the server plug-in to the PowerCenter server `javalib` directory. Follow the procedures in Copying the Plug-in Library on the Server.
Registering the Plug-in's MetaData

The PowerCenter repository needs information about the Vertica plug-in in order to enable clients to use it. This information is supplied in an XML-format file named vertica.xml. Perform the following to register the plug-in's metadata.

Switch to Exclusive Mode

Before you can register the plug-in's metadata, you must logon to PowerCenter and switch to exclusive mode to ensure that the repository does not change while you are registering the plug-in.

1. Place or copy the file vertica.xml to your system.

2. Open a browser and log into the PowerCenter domain's Administration Console.

3. Select the Domain tab, and click Services and Nodes.
4. In the Domain Navigator, click the entry for the repository that you want to connect to Vertica.

5. Under the Properties tab, click Edit to edit the Repository Properties section.

6. In the Operating Mode list box, choose Exclusive, and then click OK.
7. In the Restart Service prompt, click OK to confirm switching to exclusive mode.

The Repository Service may take from a few moments to several minutes to restart and re-enable itself.

8. Wait until you see the notice, This PowerCenter Repository Service is available.
Register the Plug-in

1. On the Plug-ins tab, click the icon for registering a plug-in.

2. On the Choose a plug-in field, click Browse and select the vertica.xml in the folder where you earlier placed the file.

3. Under the Repository Authentication section, enter your repository username and password.

   Note: If you are upgrading from a previously installed plug-in version, select the checkbox, Update existing plug-in registration. Otherwise leave the box cleared.

4. Click OK to upload the metadata file. The PowerCenter Administration Console uploads the metadata file and registers the Vertica plug-in data.
Switch Back to Normal Mode


2. In the Operating Mode list box, choose Normal.

3. In the Restart Service prompt, click OK to confirm switching to normal mode.

   The Repository Service may take from a few moments to several minutes to restart and re-enable itself.

Adding the Library File to the Windows Registry

For each PowerCenter client system that you want to use with Vertica, you must install a copy of the verticacli9.dll file in the client binary folder. This folder is named client\bin in the PowerCenter install directory. The following path is typical of a PowerCenter installation for Informatica version 9.5.1:

C:\Informatica\9.5.1\clients\PowerCenterClient\client\bin

Copy and Register the verticacli9.dll

Copy the library file to the client binary directory (i.e., client\bin).

Then, add a registry entry to the Windows registry. Adding this entry tells the PowerCenter Designer to load the plug-in library. Perform one of the following to register the plug-in library.

Note: The registry file is specific to Informatica PowerCenter version 9.5.1. The Vertica Plug-in for Informatica has only been tested with this version. If you want to try to use it with another version of PowerCenter, you will need to manually add configuration information to the Windows registry, as explained below.

Register the verticacli9.dll Using a Registry File for Informatica PowerCenter 9.5.x
1. Double-click the registry file in Windows Explorer:
   - Use vertica-infa951-x86.reg to register the dll on 32 bit machines.
   - Use vertica-infa951-x64.reg to register the dll on 64 bit machines.

2. When asked if you want to add the contents of the file to the registry, click Yes.

Register the verticacli9.dll Manually for all Other Versions of Informatica (9.6.x, 10.x, et. al.)

1. Start the registry editor by typing regedit.exe in the Windows Start menu's command run command box.

2. Navigate to the correct location in the registry.
   - For 32-bit versions of Windows:
     ```plaintext
     HKEY_LOCAL_MACHINE\SOFTWARE\Informatica\PowerMart Client Tools\x.x.x\Plugins\Informatica
     ```
   - For 64-bit versions of Windows:
     ```plaintext
     HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Informatica\PowerMart Client Tools\x.x.x\Plugins\Informatica
     ```
   Where x.x.x is the version of Informatica you are using (for example 9.5.1).

3. Right-click in the right pane of the Registry Editor window. Select New then select String Value.

4. Change the name of the string value from New Value #1 to VERTICA.

5. Double-click the new VERTICA entry. When prompted for a new value, enter verticacli9.dll.

6. Exit the registry editor.

**Copying the Plug-in Library on the Server**

The final step in setting up the Vertica plug-in for Informatica is to copy the Vertica server-side plug-in file to the proper directory on the PowerCenter server.
vertica-informatica-plugin-server.jar (for both 32 and 64 bit servers)

Copy the library file to your server's binary directory, which is the `\bin\javalib` subdirectory in the PowerCenter server install directory. The full path to this directory for Windows is usually:

```
C:\Informatica\9.5.1\server\bin\javalib
```

Note: In addition to the file `vertica-informatica-plugin-server.jar`, you must also have the appropriate JDBC driver installed in the same directory (`/bin/javalib`). The JDBC driver you install must match your version of Vertica.

The following sample screen shows a typical Windows path for these files.

For all supported server operating systems (Linux, Solaris, AIX, HP-UX), you download and install the same file, `vertica-informatica-plugin-server.jar`.

Copy the file to the `server/bin/javalib` subdirectory of the directory where PowerCenter is installed.

**Unregistering the Plug-in**

Perform this procedure only if you need to unregister the plug-in.
1. Follow the procedure, Switch to Exclusive Mode.

2. Click the trashcan icon to unregister a plug-in.

3. In the Unregister plug-in pop-up, enter your Username and Password to unregister the plug-in, and then click OK.

4. Follow the procedure, Switch Back to Normal Mode.

**Updating the Plug-in**

Follow this procedure only if you need to update the plug-in.

1. Follow the procedure, Switch to Exclusive Mode.

2. On the Plug-ins tab, click the icon for registering a plug-in.

3. On the Choose a plug-in field, click Browse and select the location of the `vertica.xml` file.

4. Check the box, Update existing plug-in registration.

5. Enter your Username and Password to unregister the plug-in, and then click OK.
6. Follow the procedure, Switch Back to Normal Mode.
Using the Vertica Plug-in with Informatica PowerCenter

Once you have installed the Vertica Plug-in for Informatica, you can use Vertica as a source or target in Informatica PowerCenter.

The simple examples in this section walk you through importing source and target, mapping, configuring, and starting your workflow.

Importing a Source Database Table

Note: Set up a DSN for your Vertica database before you perform the procedures that follow.

The following example that shows how to import a source table from a Vertica database.

1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica source.

2. Click the Source Analyzer icon.

3. From the Sources list box, select Import from Database.
4. From the Import Tables dialog box, choose the name of your ODBC data source, and enter Username and Password.

5. Click Connect. Under Select tables, choose the schema public.
6. Choose a table and Click OK. For this example, choose only the table, datatypes_src. (You can choose a number of tables.)

The table appears in the Source Analyzer panel.

7. Change the table's database type to VERTICA. To do so, double-click the name of the table to launch the Edit Tables dialog box.

8. From the Table tab, Database type list box, choose VERTICA.

   Note: As of Informatica Version 9.6.1 Hot Fix 2 and later, if you want to use the version of Vertica you downloaded and installed, select the Vertica option from the dropdown menu. If you intend to use the new Vertica Connector from Informatica, select VERTICA from the dropdown menu.

9. Click OK. You have imported a source table. Next, follow the procedure in, Importing a Target Database Table.
Importing a Target Database Table

Note: Set up a DSN for your Vertica database before you perform the procedures that follow.

The following example shows how to import a target table from a Vertica database.

1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica target.

2. Click the Target Designer icon.

3. From the Targets list box, choose Import from Database.
4. From the Import Tables dialog box, choose the name of your ODBC data source, and enter Username and Password.

5. Click Connect. Under Select tables, choose the schema public.
6. Choose a table and Click OK. For this example, choose only the table, `datatypes_tgt`.

The table appears in the Target Designer panel.

7. Change the table's database type to VERTICA. Double-click the name of the table to launch the Edit Tables dialog box.

8. From the Table tab, Database type list box, choose VERTICA.

   **Note:** As of Informatica Version 9.6.1 Hot Fix 2 and later, if you want to use the version of Vertica you downloaded and installed, select the Vertica option from the dropdown menu. If you intend to use the new Vertica Connector from Informatica, select VERTICA from the dropdown menu.

9. Click OK. You have imported a target table. Next, follow the procedure in, **Mapping Between Source and Target Tables**.

### Mapping Between Source and Target Tables

Perform this procedure to create mapping between source and target tables.
1. In Informatica PowerCenter Designer, select the folder in the repository where you want to create your Vertica mapping.

2. Click the Mapping Designer icon.

3. From the Mappings list box, select Create.

4. Enter a mapping name and click OK.

5. Choose the source, datatypes_src, and drag it to the Mapping Designer window. An Application Source Qualifier also appears; your source is mapped to Informatica. (This example sets up a basic workflow and is not meant to be a realistic sample. Note also that this example shows that configuration changes would be required where both source and/or target are in Vertica databases.)

6. Drag your target, datatypes_tgt, to the Mapping Designer window.

7. From the Layout list box, select Autolink by Name.

8. Confirm the from and to transformations, and click OK.
9. Save your work. Next, follow the procedure in, you create a workflow that uses your mapping. Follow the procedure in, Creating a Workflow.

Creating a Workflow

Perform this procedure to create a workflow using the table mapping you previously created.

1. In Informatica PowerCenter, click the Workflow button to launch the Workflow Manager.

2. From the Workflows list box, select Create.
3. In the Create Workflow dialog box, enter a name for your new workflow and click OK.

4. From the Tasks list box, select Create.
5. Enter a name for your new task, and click Create.

6. In the Mappings dialog box, choose the mapping to associate with the session and click OK.

7. Click Done on the Create Task dialog box.
8. In the Workflow Designer, drag your task to the right of Start.

9. Select the link tasks icon, and link Start to your task. Save your work.

Next you configure your workflow connection to your database. Follow the procedure in, Configuring Your Workflow Connections.

**Configuring Your Workflow Connections**

Perform this procedure to configure your workflow connection to your database.

1. While in Workflow Manager, from the Connections list box, select Relational.

2. From the Relational Connection Browser dialog box, select your Vertica connection object and click and click New.

3. From the Select Subtype dialog box, select VerticaConnection and click OK.
4. Fill in the details within the Connection Object Definition dialog box.

a. Fill in the Name, User Name, and Password.

b. Enter the connection string, which must have the format:
   jdbc:vertica://<ip>:<port>/<dbname>

c. In the JDBC Driver Name field, enter com.vertica.jdbc.Driver

d. If you are connecting to a database that is running Vertica Release 9.0.x, select EnableStreamingBatchInsert.

   **Important:** If you are running Vertica Release 7.x on either source or target database, enable (check off) EnableStreamingBatchInsert. Note that if you are running a previous version of Vertica you can still check EnableStreamingBatchInsert; previous versions will not experience the performance improvements, but setting the option has no detrimental impact.

e. If your Vertica database is SSL enabled, check EnableSSLCIonnction, and enter a Trust Store Path and Trust Store Password.

   **Note:** You must enter a complete path for Trust Store Path (e.g., C:\Users\Administrator\Desktop\verticastore).

f. Click OK.
Next you configure your source and target for the workflow. Follow the procedure in, Configuring Source and Target and Starting Your Workflow.

**Configuring Source and Target and Starting Your Workflow**

Perform this procedure to configure source and target for your workflow. (This procedure assumes all previous procedures have been completed.)

1. While in Workflow Designer, double-click your task to launch the Edit Tasks dialog box.

2. Click the Mapping tab.

3. Configure the source.

   a. Under the Sources folder, select your source. The VERTICA_READER is listed under Readers.
b. Click the down arrow icon to bring up the Relational Connection Browser dialog box.

c. Choose your object and click OK.

4. Configure the target.
   
a. Under the Targets folder, select your target. The VERTICA_WRITER is listed under Writers.

   b. Click the down arrow icon to bring up the Relational Connection Browser dialog.

   c. Choose your object and click OK. Save your work.

5. While in Workflow Manager, from the Workflows list box, select Start Workflow. (The Workflow Monitor opens.)
Accessing and Setting Plug-in Features

You can access the Vertica plug-in features through any workflow task. This section discusses how to access the features available through the Vertica plug-in.

Accessing VERTICA_READER Plug-in Attributes

Note that, when you use Vertica as source/reader, the plug-in supports only pass-through partitioning. When you use Vertica as target/writer, the plug-in supports pass-through partitioning and key range partitioning. (Note that the key value partitioning source should not be through the Vertica plug-in.)

1. While in Workflow Manager, double-click any task.
2. In the Edit Tasks dialog, select the Mappings tab.
3. Select your source qualifier (this example uses SQ_datatypes_src).
4. Minimize the Readers and Connections areas to focus on Properties.
Plug-in features are listed under the Properties area.

The table that follows lists and describes the plug-in attributes for source/reader.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
</table>
| Schema Name       | Schema Name allows you to override the default schema name of the mapped table. If you have only one source, you can override the schema name by simply entering the new schema name. Where you have many sources in a mapping that all go to the one source qualifier, you override schema names with the following format. Use a semicolon as separator:  
<old_table_name>=<new_schema_name>; <another_table_name>=<another_new_schema_name>  

Important: In the format given above, you do not actually enter the schema name to change the schema name. Instead, you enter the table
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name in the schema name field (to the left of the equal sign). You then provide the new schema name as given in the format above (to the right of the equal sign; the equal sign acts to set the new name). Use a semicolon as separator for multiple schema name changes.</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td><code>datatype_src=newschemaname</code></td>
</tr>
</tbody>
</table>
| Table Name         | Table Names changes are similar to Schema Name changes in regards to how you format the changes. If you have only one table name to change, you just enter the new table name.  
If you have more than one table name to change, you use the following format:  
`<old_table_name>=<new_table_name>; <another_table_name>=<another_new_table_name>`  
Example: `datatypes_src=newname;othertablename_src=newerone`  
**Important:** If using the Schema Name option along with the Table Name option, note that the Schema Name entry uses the old table name versus the replacement name you have added here in the Table Name option. An example follows.  
**Schema Name** entry:  
- `tbl1=new-schema-name`  
**Table Name** entry:  
- `tbl1=tbl2`  
That is, you would not specify `tbl2` under the **Schema Name** entry. |
| Select Distinct    | If selected, returns only distinct values.                                                                                                                                                     |
| Number of Sorted Ports | Sorts incoming data, specifying order by ports.                                                                                                                                               |
| User Defined       | Allows you to do overrides to your mapping, enabling you to do very specific
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
</table>
| Join                  | joins (e.g., inner and outer joins). When you click the arrow to the right of the field, a SQL box pops up. What you enter in the box becomes a custom join clause added to the generated SQL statement. Basically you can enter a typical join statement with SELECT being assumed. Examples:  
  * `s1.a.col = s2.b.col`  
    (where `s1` and `s2` are schema names)  
  * `a JOIN b ON a.col = b.col`  
  * `a.col = b.col` |
| Pre SQL               | Enter complete SQL statements that run before you read a table. For example, truncate or add data to a table before you read it. |
| Post SQL              | Enter complete SQL statements that run after you read a table. |
| Modulus Partitioning  | If selected, performs modulus partitioning (no replication of data) rather than straight pass-through partitioning. Check this option only if using pass-through partitioning. |
| Time UTC Conversion   | If selected, keeps time synchronized when you are using Vertica as both source and target. Time data changes to UTC time zone.  
  Select this option and the writer option only when using Vertica as both source and target.  
  If unchecked, when you are using Vertica as both source and target, time data changes to the JVM time zone. |
| SQL Query             | Overrides the entire query rather than overriding just a portion of a query. |
| Source Filter         | Allows you to do overrides to your mapping by overriding the WHERE clause (similar to the way User Defined Join allows you to override a JOIN clause). |
Accessing VERTICA_WRITER Plug-in Attributes

1. While in Workflow Manager, double-click any task.
2. In the Edit Tasks dialog box, select the Mappings tab.
3. Select your source qualifier (this example uses datatypes_tgt).
4. Minimize the Readers and Connections areas to focus on Properties.

Plug-in features are listed under the Properties area.

The table that follows lists and describes the plug-in attributes for target/writer.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Name</td>
<td>Change the schema name by entering a new name in this field.</td>
</tr>
<tr>
<td>Truncate Target Table</td>
<td>Select this option if you have a workflow that should truncate its targeted table before loading data.</td>
</tr>
<tr>
<td>Pre SQL</td>
<td>Enter complete SQL statements that run before you write to a table.</td>
</tr>
<tr>
<td>Post SQL</td>
<td>Enter complete SQL statements that run after you write to a table.</td>
</tr>
<tr>
<td>Target Table Name</td>
<td>Change the target table name by entering a new name in this field.</td>
</tr>
<tr>
<td>Copy Direct</td>
<td>When selected, writes directly to the ROS container. More efficient for bulk loading.</td>
</tr>
<tr>
<td>Reject file directory</td>
<td>Reject file directory and Reject file name work in tandem to record rejected rows. In this field you specify the directory path for the reject file.</td>
</tr>
<tr>
<td>Reject file name</td>
<td>Specify the name of the file that holds the rejected rows. There is one log file per partition. If there are multiple log files, a number is appended to the file names. For example, rejects.txt would become rejects_01.txt and rejects_02.txt.</td>
</tr>
<tr>
<td>Insert</td>
<td>If selected, makes insert the update strategy for the target.</td>
</tr>
<tr>
<td>Delete</td>
<td>If selected, makes delete the update strategy for the target.</td>
</tr>
<tr>
<td>Update</td>
<td>List box offers standard update options for target.</td>
</tr>
<tr>
<td>Time UTC Conversion</td>
<td>If selected, keeps time synchronized when you are using Vertica as both source and target. Time data changes to UTC time zone.</td>
</tr>
<tr>
<td></td>
<td>Select this option and the reader option only when using Vertica as both source and target.</td>
</tr>
<tr>
<td></td>
<td>If unchecked, when you are using Vertica as both source and target, time data changes to the JVM time zone.</td>
</tr>
<tr>
<td>Copy Local Method</td>
<td>Choose method from list box.</td>
</tr>
<tr>
<td></td>
<td><strong>None.</strong> The default; in this case, no local copy method is used to stream data.</td>
</tr>
</tbody>
</table>

**Important:** To take advantage of increased performance of
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableStreamingBatchInsert</td>
<td>you must set Copy Local Method to None.</td>
</tr>
<tr>
<td>Delimited</td>
<td>Creates a stream with pipes and newline delimiters. This method fails if there are pipes or newline delimiters in the data you are transmitting.</td>
</tr>
<tr>
<td>Native Varchar</td>
<td>Converts to new format. Use if data is mostly some form of strings.</td>
</tr>
<tr>
<td>Native binary</td>
<td>Serializes java objects to Vertica objects.</td>
</tr>
</tbody>
</table>
| Target Number             | **Default behavior.** The Target Number default is zero. If you leave the default (or set Target Number to any number less than or equal to 1), the plug-in uses the IP address you specify in the Connection String when configuring workflow connections. See Configuring Your Workflow Connections for more information on where you specify the IP address. In regards to load balancing, the Target Number default operates as follows.  
  * If the IP Address you specified in the Connection String is not a Vertica node, the plug-in targets that IP Address. Any load balancing policy enabled on the IP Address is used. 
  * If the node you specified in the Connection String is a Vertica node, note the following:
    
    Vertica native connection load balancing is off by default; in this case, the plug-in uses the IP address you specify in the connection string. 
    
    If Vertica load balancing is enabled, then the plug-in targets nodes as defined by the load balancing scheme on the Vertica node.  
  **Non-default behavior.** You override the plug-in’s default behavior by setting the option Target Number to a value greater than 1 (up to 16). When the value of the Target Number is set to greater than 1, the plug-in targets that number of Vertica nodes and uses ROUNDROBIN as its load balancing scheme. This overrides the native Vertica load balancing scheme. 
  For information on native connection load balancing, refer to About About Native Connection Load Balancing in the Administrator’s Guide. For information on setting a load balancing policy on a Vertica server, refer to SET_LOAD_BALANCE_POLICY in the SQL Reference Manual. |
Setting EnableStreamingBatchInsert

If you are connecting to a database that is running Vertica Release 9.0.x, for best performance, you should always implement EnableStreamingBatchInsert.

If you are running an earlier version of Vertica, setting EnableStreamingBatchInsert has no impact on performance.

Find the full procedure for accessing the setting in the section, Configuring Your Workflow Connections. For a workflow that is already set up:

1. While in Workflow Manager, select a task.
2. From the Connections list box, select Relational.
3. Click Edit.
4. In the Connection Object Definition dialog box, check off EnableStreamingBatchInsert.
5. Click OK. Save your work.

Important: To take advantage of increased performance of EnableStreamingBatchInsert, you must set Copy Local Method to None.

Enabling SSL

If your Vertica database is SSL enabled, then within the Connection Object Definition dialog, check EnableSSLConnection, and enter a Trust Store Path and Trust Store Password.

Note: You must enter a complete path for Trust Store Path (e.g., C:\Users\Administrator\Desktop\verticastore).

For a simple example of where to access the SSL setting, refer to Configuring Your Workflow Connections.
Best Practices

This section includes memory requirement considerations and other tips.

Setting Memory Properties for a Task

OpenText recommends that you increase memory allocation to improve performance.

Note: The default buffer size for Informatica PowerCenter is set very conservatively. These settings can cause PowerCenter to send Vertica many small batches, rather than a few large batches. The overhead of these many small batches can cause loading performance issues. To resolve these performance issues, you should change PowerCenter's batch size settings. Your specific settings depend upon your system resources and needs.

Perform the following procedure to increase memory allocation for a task.

1. While in Workflow Manager, double-click the task that connects to Vertica.
2. In the Edit Tasks dialog box, under the Mapping tab, select Memory Properties. Considering your task requirements and your system limitations, set the following attributes.

   Note: Allocate more memory than mentioned here according to your system limitations and needs. The settings given may not be realistic for the tasks you intend to perform.

   a. Set Default buffer block size to at least 16 MB.
   b. Set Maximum Memory Allowed for Auto Memory Attributes to at least 512 MB.
   c. Set Maximum Percentage of Total Memory Allowed for Auto Memory Attributes to
Setting JVM Memory Properties

OpenText recommends the following settings for your JVM (Java Virtual Machine).

1. Log on to the PowerCenter domain’s Administration Console.
2. Select the Domain tab, and click Services and Nodes.
3. In the Domain Navigator, click the entry for the PowerCenter Integration Service.
4. Click Edit next to the Custom Properties section.
5. In the Edit Custom Properties dialog box, click New.
   a. Enter a name.
   b. Set a minimum heap memory size to at least 1024 m. Enter `-Xms=1024m`
   c. Click OK.

6. In the Edit Custom Properties dialog box, click New again.
   a. Enter a name.
   b. Set a maximum heap memory size to at least double the minimum you just entered.
      Enter `-Xmx=2048m`
c. Click OK.

Communicating when Informatica and Vertica Are on Different Networks

This best practice concerns communication with a Vertica cluster from Informatica when Informatica and Vertica are on separate networks. If Informatica and Vertica are on the same network, you do not need to implement the changes described here.

If Informatica and Vertica are on separate networks:

- Set up a public network for import/export and specify an export address for your individual nodes. See Using Public and Private IP Networks in the Administrator’s Guide, specifically the section, Identify the Database or Nodes Used for Import/Export. See also ALTER DATABASE and ALTER NODE in the SQL Reference manual for subnet and node-related tasks.
The Vertica plug-in for Informatica accesses the EXPORT_ADDRESS column of the V_Catalog schema. (For information on viewing the EXPORT_ADDRESS column, see the SQL Reference manual, specifically the section NODES in the V_Catalog Schema.)

- Once the public network is set-up properly, and an export address is assigned to each node, Informatica can then read and write to a Vertica cluster on a different network.

**Important:** If you are using the Vertica plug-in for Informatica on Vertica Release 6.x, the Informatica user must have the PSEUDOSUPERUSER role to access the export addresses on the different network. For general information on the PSEUDOSUPERUSER role, see the Administrator's Guide, PSEUDOSUPERUSER Role. If you are using Vertica Release 7.0.x, the Informatica user does not need the PSEUDOSUPERUSER role.

---

**Modifying powrmart.ini**

When reading or importing a table, you can receive a pop-up warning, such as the following, concerning a missing DLL.

![Designer](image)

You can make a correction so that the warning no longer appears by adding the line `Vertica Database=PMODBC.DLL` to the powrmart.ini file in the section ODBCDLL. See the following example.

1. Using the Import Tables option in the Informatica PowerCenter Designer, the system displays a pop-up warning about a missing DLL. Click OK.
2. The system displays another pop-up letting you know that Informatica is using EXTODBC.DLL to support Vertica. Click OK.
3. Open the powrmart.ini file for editing.

4. Add the following line in the section ODBCDLL.

   Vertica Database=PMODBC.DLL

---

Deleting Records on a Target Table

Follow these notes to delete records on a target table.

1. On the target table, you must have defined a primary key. From Edit Tasks, select the Properties tab, and, under General Options, set Treat Source row as to Delete.
2. Also on the target table, from the Edit Tasks dialog, choose the Mapping tab. Under Properties, check-off Delete.
Welcome to the Vertica Error Codes guide. This guide is mainly for developers who need to understand how Vertica's error codes relate to error states returned by the ODBC and JDBC drivers.
The Different Ways Vertica Reports Errors

Vertica reports warnings and errors via two different mechanisms: SQLSTATEs and error messages. SQLSTATEs are intended for use by client applications, such as those accessing Vertica via ODBC or JDBC. Error messages are displayed to interactive users (for example, users connected to Vertica through vsqI) and written to error logs.
About SQLSTATE

Vertica reports the success or failure of each statement it executes to client applications using a five-character SQLSTATE value. Many of these values are defined by the SQL standard. Others (identified by the letter "V" in their values) are Vertica-specific.

SQLSTATE values are grouped into classes which are defined by the first two characters in the SQLSTATE value. The last three characters indicate a specific condition within a class. For example, the SQLSTATE class 22 represents all data errors. The specific SQLSTATE value 22012 represents a division by zero error. SQLSTATE classes let an application that does not recognize a specific SQLSTATE value to still get a general idea of the result.
Warning and Error Messages

Each error and warning message displayed to interactive users or written to a log file by Vertica has its own numeric error code assigned to it. For example:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR 3117</td>
<td>Division by zero</td>
</tr>
<tr>
<td>WARNING 4098</td>
<td>No projections found</td>
</tr>
<tr>
<td>ERROR 5617</td>
<td>Multiple WITH clauses not allowed</td>
</tr>
</tbody>
</table>

The error code number is not related to the SQLSTATE value. However, error and warning messages do correspond to a specific SQLSTATE. They are just a more-specific human-readable message compared to the SQLSTATE, which is mainly intended for client applications.

For example, all warning messages displayed by Vertica correspond to the SQLSTATE class 01. The warning message "WARNING 3084: Design Workspace couldn't be dropped" corresponds to the SQLSTATE value 01000 ERRCODE_WARNING.

Error codes do not change between Vertica releases, but individual error and warning messages may be added or removed in new releases. Your client application should not depend on particular error code appearing from one release to the next. Instead, it should use the SQLSTATE value to determine the result of executing a statement.

See the SQL State List for a list of all the SQLSTATE classes and values defined by Vertica. This table also links to lists of error or warning messages that are associated with each SQLSTATE value.
# SQL State List

The following table lists the SQLSTATE classes and individual SQLSTATE codes.

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class 00—Successful Completion</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td>ERRCODE_SUCCESSFUL_COMPLETION</td>
<td></td>
</tr>
<tr>
<td><strong>Class 01—Warning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>ERRCODE_WARNING</td>
<td>associated warning messages</td>
</tr>
<tr>
<td>01003</td>
<td>ERRCODE_WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION</td>
<td></td>
</tr>
<tr>
<td>01004</td>
<td>ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION</td>
<td>associated warning messages</td>
</tr>
<tr>
<td>01006</td>
<td>ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED</td>
<td>associated warning messages</td>
</tr>
<tr>
<td>01007</td>
<td>ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED</td>
<td>associated warning messages</td>
</tr>
<tr>
<td>01008</td>
<td>ERRCODE_WARNING_PRIVILEGE_ALREADY_GRANTED</td>
<td></td>
</tr>
<tr>
<td>01009</td>
<td>ERRCODE_WARNING_PRIVILEGE_ALREADY_REVOKED</td>
<td></td>
</tr>
<tr>
<td>0100C</td>
<td>ERRCODE_WARNING_DYNAMIC_RESULT_SETS_RETURNED</td>
<td></td>
</tr>
<tr>
<td>01V01</td>
<td>ERRCODE_WARNING_DEPRECATED_FEATURE</td>
<td>associated warning messages</td>
</tr>
<tr>
<td>01V02</td>
<td>ERRCODE_WARNING_QUERY_RETRIEVED</td>
<td></td>
</tr>
<tr>
<td><strong>Class 02—No Data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02000</td>
<td>ERRCODE_NO_DATA</td>
<td></td>
</tr>
<tr>
<td>02001</td>
<td>ERRCODE_NO_ADDITIONAL_DYNAMIC_RESULT_SETS_RETURNED</td>
<td></td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>Class 03—SQL Statement Not Yet Complete</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03000</td>
<td>ERRCODE_SQL_STATEMENT_NOT_YET_COMPLETE</td>
<td></td>
</tr>
<tr>
<td><strong>Class 08—Client Connection Exception</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08000</td>
<td>ERRCODE_CONNECTION_EXCEPTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>08001</td>
<td>ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>08003</td>
<td>ERRCODE_CONNECTION_DOES_NOT_EXIST</td>
<td>associated error messages</td>
</tr>
<tr>
<td>08004</td>
<td>ERRCODE_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION</td>
<td></td>
</tr>
<tr>
<td>08006</td>
<td>ERRCODE_CONNECTION_FAILURE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>08007</td>
<td>ERRCODE_TRANSACTION_RESOLUTION_UNKNOWN</td>
<td></td>
</tr>
<tr>
<td>08V01</td>
<td>ERRCODE_PROTOCOL_VIOLATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td><strong>Class 09—Triggered Action Exception</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09000</td>
<td>ERRCODE_TRIGGERED_ACTION_EXCEPTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td><strong>Class 0A—Feature Not Supported</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0A000</td>
<td>ERRCODE_FEATURE_NOT_SUPPORTED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>0A005</td>
<td>ERRCODE_PLAN_TO_SQL_NOT_SUPPORTED</td>
<td>associated error messages</td>
</tr>
<tr>
<td><strong>Class 0B—Invalid Transaction Initiation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B000</td>
<td>ERRCODE_INVALID_TRANSACTION_INITIATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>0F000</td>
<td>ERRCODE_LOCATOR_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>0F001</td>
<td>ERRCODE_L_E_INVALID_SPECIFICATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Class 0L—Invalid Grantor</strong></td>
<td></td>
</tr>
<tr>
<td>0L000</td>
<td>ERRCODE_INVALID_GRANTOR</td>
<td></td>
</tr>
<tr>
<td>0LV01</td>
<td>ERRCODE_INVALID_GRANT_OPERATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td></td>
<td><strong>Class 0P—Invalid Role Specification</strong></td>
<td></td>
</tr>
<tr>
<td>0P000</td>
<td>ERRCODE_INVALID_ROLE_SPECIFICATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Class 21—Cardinality Violation</strong></td>
<td></td>
</tr>
<tr>
<td>21000</td>
<td>ERRCODE_CARDINALITY_VIOLATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Class 22—Data Exception</strong></td>
<td></td>
</tr>
<tr>
<td>22000</td>
<td>ERRCODE_DATA_EXCEPTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22001</td>
<td>ERRCODE_STRING_DATA_RIGHT_TRUNCATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22002</td>
<td>ERRCODE_NULL_VALUE_NO_INDICATOR_PARAMETER</td>
<td></td>
</tr>
<tr>
<td>22003</td>
<td>ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22004</td>
<td>ERRCODE_NULL_VALUE_NOT_ALLOWED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22005</td>
<td>ERRCODE_ERROR_IN_ASSIGNMENT</td>
<td></td>
</tr>
<tr>
<td>22007</td>
<td>ERRCODE_INVALID_DATETIME_FORMAT</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22008</td>
<td>ERRCODE_DATETIME_FIELD_OVERFLOW</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>22009</td>
<td>ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>2200B</td>
<td>ERRCODE_ESCAPE_CHARACTER_CONFLICT</td>
<td>associated error messages</td>
</tr>
<tr>
<td>2200C</td>
<td>ERRCODE_INVALID_USE_OF_ESCAPE_CHARACTER</td>
<td></td>
</tr>
<tr>
<td>2200D</td>
<td>ERRCODE_INVALID_ESCAPE_OCTET</td>
<td>associated error messages</td>
</tr>
<tr>
<td>2200F</td>
<td>ERRCODE ZERO LENGTH CHARACTER STRING</td>
<td></td>
</tr>
<tr>
<td>2200G</td>
<td>ERRCODE_MOST_SPECIFIC_TYPE_MISMATCH</td>
<td></td>
</tr>
<tr>
<td>22010</td>
<td>ERRCODE_INVALID_INDICATOR_PARAMETER_VALUE</td>
<td></td>
</tr>
<tr>
<td>22011</td>
<td>ERRCODE_SUBSTRING_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22012</td>
<td>ERRCODE_DIVISION_BY_ZERO</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22015</td>
<td>ERRCODE_INTERVAL_FIELD_OVERFLOW</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22018</td>
<td>ERRCODE_INVALID_CHARACTER_VALUE_FOR_CAST</td>
<td></td>
</tr>
<tr>
<td>22019</td>
<td>ERRCODE_INVALID_ESCAPE_CHARACTER</td>
<td>associated error messages</td>
</tr>
<tr>
<td>2201B</td>
<td>ERRCODE_INVALID_REGULAR_EXPRESSION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>2201E</td>
<td>ERRCODE_INVALID_ARGUMENT_FOR_LOG</td>
<td></td>
</tr>
<tr>
<td>2201F</td>
<td>ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION</td>
<td></td>
</tr>
<tr>
<td>2201G</td>
<td>ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22020</td>
<td>ERRCODE_INVALID_LIMIT_VALUE</td>
<td></td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>22021</td>
<td>ERRCODE_CHARACTER_NOT_IN_REPERTOIRE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22022</td>
<td>ERRCODE_INDICATOR_OVERFLOW</td>
<td></td>
</tr>
<tr>
<td>22023</td>
<td>ERRCODE_INVALID_PARAMETER_VALUE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22024</td>
<td>ERRCODE_UNTERMINATED_C_STRING</td>
<td></td>
</tr>
<tr>
<td>22025</td>
<td>ERRCODE_INVALID_ESCAPE_SEQUENCE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22026</td>
<td>ERRCODE_STRING_DATA_LENGTH_MISMATCH</td>
<td></td>
</tr>
<tr>
<td>22027</td>
<td>ERRCODE_TRIM_ERROR</td>
<td></td>
</tr>
<tr>
<td>2202E</td>
<td>ERRCODE_ARRAY_ELEMENT_ERROR</td>
<td></td>
</tr>
<tr>
<td>22906</td>
<td>ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V01</td>
<td>ERRCODE_FLOATING_POINT_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>22V02</td>
<td>ERRCODE_INVALID_TEXT_REPRESENTATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V03</td>
<td>ERRCODE_INVALID_BINARY_REPRESENTATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V04</td>
<td>ERRCODE_BAD_COPY_FILE_FORMAT</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V05</td>
<td>ERRCODE_UNTRANSLATABLE_CHARACTER</td>
<td></td>
</tr>
<tr>
<td>22V08</td>
<td>ERRCODE_ESCAPE_CHARACTER_ON_NOESCAPE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V21</td>
<td>ERRCODE_INVALID_EPOCH</td>
<td>associated error messages</td>
</tr>
<tr>
<td>22V22</td>
<td>ERRCODE_PLPGSQL_ERROR</td>
<td></td>
</tr>
<tr>
<td>22V23</td>
<td>ERRCODE_RAISE_EXCEPTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>22V24</td>
<td>ERRCODE_COPY_PARSE_ERROR</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

### Class 23—Integrity Constraint Violation

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>23000</td>
<td>ERRCODE_INTEGRITY_CONSTRAINT_VIOLATION</td>
<td></td>
</tr>
<tr>
<td>23001</td>
<td>ERRCODE_RESTRICT_VIOLATION</td>
<td></td>
</tr>
<tr>
<td>23502</td>
<td>ERRCODE_NOT_NULL_VIOLATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>23503</td>
<td>ERRCODE_FOREIGN_KEY_VIOLATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>23505</td>
<td>ERRCODE_UNIQUE_VIOLATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>23514</td>
<td>ERRCODE_CHECK_VIOLATION</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

### Class 24—Invalid Cursor State

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>ERRCODE_INVALID_CURSOR_STATE</td>
<td></td>
</tr>
</tbody>
</table>

### Class 25—Invalid Transaction State

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>25000</td>
<td>ERRCODE_INVALID_TRANSACTION_STATE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>25001</td>
<td>ERRCODE_ACTIVE_SQL_TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>25002</td>
<td>ERRCODE_BRANCH_TRANSACTION_ALREADY_ACTIVE</td>
<td></td>
</tr>
<tr>
<td>25003</td>
<td>ERRCODE_INAPPROPRIATE_ACCESS_MODE_FOR_BRANCH_TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>25004</td>
<td>ERRCODE_INAPPROPRIATE_ISOLATION_LEVEL_FOR_BRANCH_TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>25005</td>
<td>ERRCODE_NO_ACTIVE_SQL_TRANSACTION_FOR_BRANCH_TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>25006</td>
<td>ERRCODE_READ_ONLY_SQL_TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>25007</td>
<td>ERRCODE_SCHEMA_AND_DATA_STATEMENT_MIXING_NOT_SUPPORTED</td>
<td></td>
</tr>
<tr>
<td>25008</td>
<td>ERRCODE HELD_CURSOR.Requires SAME ISOLATION LEVEL</td>
<td></td>
</tr>
<tr>
<td>25V01</td>
<td>ERRCODE_NO_ACTIVE_SQL_TRANSACTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>25V02</td>
<td>ERRCODE_IN_FAILED_SQL_TRANSACTION</td>
<td></td>
</tr>
</tbody>
</table>

**Class 26—Invalid SQL Statement Name**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26000</td>
<td>ERRCODE_INVALID_SQL_STATEMENT_NAME</td>
</tr>
</tbody>
</table>

**Class 27—Triggered Data Change Violation**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27000</td>
<td>ERRCODE_TRIGGERED_DATA_CHANGE_VIOLATION</td>
</tr>
</tbody>
</table>

**Class 28—Invalid Authorization Specification**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>28000</td>
<td>ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>28001</td>
<td>ERRCODE_ACCOUNT_LOCKED</td>
<td></td>
</tr>
<tr>
<td>28002</td>
<td>ERRCODE_PASSWORD_EXPIRED</td>
<td></td>
</tr>
<tr>
<td>28003</td>
<td>ERRCODE_PASSWORD_IN_GRACE_PERIOD</td>
<td></td>
</tr>
</tbody>
</table>

**Class 2B—Dependent Privilege Descriptors Still Exist**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B000</td>
<td>ERRCODE_DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST</td>
<td></td>
</tr>
<tr>
<td>2BV01</td>
<td>ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 2D—Invalid Transaction Termination**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D000</td>
<td>ERRCODE_INVALID_TRANSACTION_TERMINATION</td>
</tr>
</tbody>
</table>

**Class 2F—SQL Routine Exception**
<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2F000</td>
<td>ERRCODE_SQL_ROUTINE_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>2F002</td>
<td>ERRCODE_S_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED</td>
<td></td>
</tr>
<tr>
<td>2F003</td>
<td>ERRCODE_S_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED</td>
<td></td>
</tr>
<tr>
<td>2F004</td>
<td>ERRCODE_S_R_E_READING_SQL_DATA_NOT_PERMITTED</td>
<td></td>
</tr>
<tr>
<td>2F005</td>
<td>ERRCODE_S_R_E_FUNCTION_EXECUTED_NO_RETURN_STATEMENT</td>
<td></td>
</tr>
</tbody>
</table>

**Class 34—Invalid Cursor Name**

| 34000 | ERRCODE_INVALID_CURSOR_NAME |        |

**Class 38—External Routine Exception**

| 38000 | ERRCODE_EXTERNAL_ROUTINE_EXCEPTION |        |
| 38001 | ERRCODE_E_R_E_CONTAINING_SQL_NOT_PERMITTED |        |
| 38002 | ERRCODE_E_R_E_MODIFYING_SQL_DATA_NOT_PERMITTED |        |
| 38003 | ERRCODE_E_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED |        |
| 38004 | ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED |        |

**Class 39—External Routine Invocation Exception**

| 39000 | ERRCODE_EXTERNAL_ROUTINE_INVOCATION_EXCEPTION |        |
| 39001 | ERRCODE_E_R_I_E_INVALID_SQLSTATE_RETURNED |        |
| 39004 | ERRCODE_E_R_I_E_NULL_VALUE_NOT_ALLOWED |        |
| 39V01 | ERRCODE_E_R_I_E_TRIGGER_PROTOCOL_VIOLATED |        |

*associated error messages*
<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>39V02</td>
<td>ERRCODE_E_R_I_E_SRF_PROTOCOL_VIOLATED</td>
<td></td>
</tr>
</tbody>
</table>

**Class 3B—Savepoint Exception**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>3B000</td>
<td>ERRCODE_SAVEPOINT_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>3B001</td>
<td>ERRCODE_S_E_INVALID_SPECIFICATION</td>
<td></td>
</tr>
</tbody>
</table>

**Class 3D—Invalid Catalog Name**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D000</td>
<td>ERRCODE_INVALID/catalog_name</td>
<td></td>
</tr>
</tbody>
</table>

**Class 3F—Invalid Schema Name**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F000</td>
<td>ERRCODE_INVALID/SCHEMA_NAME</td>
<td></td>
</tr>
</tbody>
</table>

**Class 40—Transaction Rollback**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>40000</td>
<td>ERRCODE_TRANSACTION_ROLLBACK</td>
<td></td>
</tr>
<tr>
<td>40001</td>
<td>ERRCODE_T_R_SERIALIZATION_FAILURE</td>
<td></td>
</tr>
<tr>
<td>40002</td>
<td>ERRCODE_T_R_INTEGRITY_CONSTRAINT_VIOLATION</td>
<td></td>
</tr>
<tr>
<td>40003</td>
<td>ERRCODE_T_R_STATEMENT_COMPLETION_UNKNOWN</td>
<td></td>
</tr>
<tr>
<td>40V01</td>
<td>ERRCODE_T_R_DEADLOCK_DETECTED</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 42—Syntax Error or Access Rule Violation**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>42000</td>
<td>ERRCODE_SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION</td>
<td></td>
</tr>
<tr>
<td>42501</td>
<td>ERRCODE_INSUFFICIENT_PRIVILEGE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42601</td>
<td>ERRCODE_SYNTAX_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42602</td>
<td>ERRCODE_INVALID_NAME</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42611</td>
<td>ERRCODE_INVALID_COLUMN_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>42622</td>
<td>ERRCODE_NAME_TOO_LONG</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42701</td>
<td>ERRCODE_DUPLICATE_COLUMN</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42702</td>
<td>ERRCODE_AMBIGUOUS_COLUMN</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42703</td>
<td>ERRCODE_UNDEFINED_COLUMN</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42704</td>
<td>ERRCODE_UNDEFINED_OBJECT</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42710</td>
<td>ERRCODE_DUPLICATE_OBJECT</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42712</td>
<td>ERRCODE_DUPLICATE_ALIAS</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42723</td>
<td>ERRCODE_DUPLICATE_FUNCTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42725</td>
<td>ERRCODE_AMBIGUOUS_FUNCTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42803</td>
<td>ERRCODE_GROUPING_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42804</td>
<td>ERRCODE_DATATYPE_MISMATCH</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42809</td>
<td>ERRCODE_WRONG_OBJECT_TYPE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42830</td>
<td>ERRCODE_INVALID_FOREIGN_KEY</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42846</td>
<td>ERRCODE_CANNOT_COERCE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42883</td>
<td>ERRCODE_UNDEFINED_FUNCTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>42939</td>
<td>ERRCODE_RESERVED_NAME</td>
<td>messages</td>
</tr>
<tr>
<td>42P20</td>
<td>ERRCODE_WINDOWING_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V01</td>
<td>ERRCODE_UNDEFINED_TABLE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V02</td>
<td>ERRCODE_UNDEFINED_PARAMETER</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V03</td>
<td>ERRCODE_DUPLICATE_CURSOR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V04</td>
<td>ERRCODE_DUPLICATE_DATABASE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V05</td>
<td>ERRCODE_DUPLICATE_PSTATEMENT</td>
<td></td>
</tr>
<tr>
<td>42V06</td>
<td>ERRCODE_DUPLICATE_SCHEMA</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V07</td>
<td>ERRCODE_DUPLICATE_TABLE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V08</td>
<td>ERRCODE_AMBIGUOUS_PARAMETER</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V09</td>
<td>ERRCODE_AMBIGUOUS_ALIAS</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V10</td>
<td>ERRCODE_INVALID_COLUMN_REFERENCE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V11</td>
<td>ERRCODE_INVALID_CURSOR_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V12</td>
<td>ERRCODE_INVALID_DATABASE_DEFINITION</td>
<td></td>
</tr>
<tr>
<td>42V13</td>
<td>ERRCODE_INVALID_FUNCTION_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>42V14</td>
<td>ERRCODE_INVALID_PSTATEMENT_DEFINITION</td>
<td></td>
</tr>
<tr>
<td>42V15</td>
<td>ERRCODE_INVALID_SCHEMA_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V16</td>
<td>ERRCODE_INVALID_TABLE_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V17</td>
<td>ERRCODE_INVALID_OBJECT_DEFINITION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V18</td>
<td>ERRCODE_INDETERMINATE_DATATYPE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V21</td>
<td>ERRCODE_UNDEFINED_PROJECTION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V22</td>
<td>ERRCODE_UNDEFINED_NODE</td>
<td></td>
</tr>
<tr>
<td>42V23</td>
<td>ERRCODE_UNDEFINED_PERMUTATION</td>
<td></td>
</tr>
<tr>
<td>42V24</td>
<td>ERRCODE_UNDEFINED_USER</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V25</td>
<td>ERRCODE_PATTERN_MATCH_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>42V26</td>
<td>ERRCODE_DUPLICATE_NODE</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 44—WITH CHECK OPTION Violation**

| 44000    | ERRCODE_WITH_CHECK_OPTION_VIOLATION  |                              |

**Class 53—Insufficient Resources**

<p>| 53000    | ERRCODE_INSUFFICIENT_RESOURCES      | associated error messages    |
| 53100    | ERRCODE_DISK_FULL                   | associated error messages    |
| 53200    | ERRCODE_OUT_OF_MEMORY               | associated error messages    |</p>
<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>53300</td>
<td>ERRCODE_TOO_MANY_CONNECTIONS</td>
<td></td>
</tr>
</tbody>
</table>

**Class 54—Program Limit Exceeded**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>54000</td>
<td>ERRCODE_PROGRAM_LIMIT_EXCEEDED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>54001</td>
<td>ERRCODE_STATEMENT_TOO_COMPLEX</td>
<td>associated error messages</td>
</tr>
<tr>
<td>54011</td>
<td>ERRCODE_TOO_MANY_COLUMNS</td>
<td>associated error messages</td>
</tr>
<tr>
<td>54023</td>
<td>ERRCODE_TOO_MANY_ARGUMENTS</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 55—Object Not In Prerequisite State**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>55000</td>
<td>ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>55006</td>
<td>ERRCODE_OBJECT_IN_USE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>55V02</td>
<td>ERRCODE_CANT_CHANGE_RUNTIME_PARAM</td>
<td>associated error messages</td>
</tr>
<tr>
<td>55V03</td>
<td>ERRCODE_LOCK_NOT_AVAILABLE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>55V04</td>
<td>ERRCODE_TM_MARKER_NOT_AVAILABLE</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 57—Operator Intervention**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>57000</td>
<td>ERRCODE_OPERATOR_INTERVENTION</td>
<td></td>
</tr>
<tr>
<td>57014</td>
<td>ERRCODE_QUERY_CANCELED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>57015</td>
<td>ERRCODE_SLOW_DELETE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>57V01</td>
<td>ERRCODE_ADMIN_SHUTDOWN</td>
<td>associated error messages</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>57V02</td>
<td>ERRCODE_CRASH_SHUTDOWN</td>
<td>messages</td>
</tr>
<tr>
<td>57V03</td>
<td>ERRCODE_CANNOT_CONNECT_NOW</td>
<td>associated error messages</td>
</tr>
<tr>
<td>57V04</td>
<td>ERRCODE_DML_COMMIT_DURING_SHUTDOWN</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class 58—System Error**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>58030</td>
<td>ERRCODE_IO_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>58V01</td>
<td>ERRCODE_UNDEFINED_FILE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>58V02</td>
<td>ERRCODE_DUPLICATE_FILE</td>
<td></td>
</tr>
</tbody>
</table>

**Class V1—Vertica-specific multi-node errors class**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1001</td>
<td>ERRCODE_LOST_CONNECTIVITY</td>
<td>associated error messages</td>
</tr>
<tr>
<td>V1002</td>
<td>ERRCODE_K_SAFETY_VIOLATION</td>
<td>associated error messages</td>
</tr>
<tr>
<td>V1003</td>
<td>ERRCODE_CLUSTER_CHANGE</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class V2—Vertica-specific miscellaneous errors class**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2000</td>
<td>ERRCODE_AUTH_FAILED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>V2001</td>
<td>ERRCODE_LICENSE_ISSUE</td>
<td>associated error messages</td>
</tr>
<tr>
<td>V2002</td>
<td>ERRCODE_MOVEOUT_ABORTED</td>
<td></td>
</tr>
</tbody>
</table>

**Class VC—Configuration File Error**

<table>
<thead>
<tr>
<th>SQLState</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC001</td>
<td>ERRCODE_CONFIG_FILE_ERROR</td>
<td>associated error</td>
</tr>
<tr>
<td>SQLState</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>VC002</td>
<td>ERRCODE_LOCK_FILE_EXISTS</td>
<td>messages</td>
</tr>
</tbody>
</table>

**Class VD—DB Designer errors**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>VD001</td>
<td>ERRCODE_DESIGNER_FUNCTION_ERROR</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class VP—User procedure errors**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP000</td>
<td>ERRCODE_USER_PROC_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>VP001</td>
<td>ERRCODE_USER_PROC_EXEC_ERROR</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

**Class VX—Internal Error**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>VX001</td>
<td>ERRCODE_INTERNAL_ERROR</td>
<td>associated error messages</td>
</tr>
<tr>
<td>VX002</td>
<td>ERRCODE_DATA_CORRUPTED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>VX003</td>
<td>ERRCODE_INDEX_CORRUPTED</td>
<td>associated error messages</td>
</tr>
<tr>
<td>VX004</td>
<td>ERRCODE_PLAN_TO_SQL_INTERNAL_ERROR</td>
<td>associated error messages</td>
</tr>
</tbody>
</table>

### Warning Messages Associated with SQLSTATE 01000

This topic lists the warnings associated with the SQLSTATE 01000.

**SQLSTATE 01000 Description**

ERRCODE_WARNING
Warning messages associated with this SQLState

WARNING 2362: Cannot begin transaction; transaction is already running
WARNING 3084: Design couldn’t be dropped
WARNING 3152: Duplicate values in columns marked as UNIQUE will now be ignored for the remainder of your session or until reenable_duplicate_key_error() is called
WARNING 3372: Failed to disable profiling: string
WARNING 3373: Failed to enable profiling: string
WARNING 3539: Incorrect results are possible. Please contact Vertica Support if unsure
WARNING 3791: Invalid view string: string
WARNING 4071: NO COMMIT option will be ignored for external table "string"
WARNING 4088: No new valid default roles specified. Retaining previous set of default roles for user string
WARNING 4098: No projections found
WARNING 4102: No rows are inserted into table "string"."string" because ON COMMIT DELETE ROWS is the default for create temporary table
WARNING 4116: No super projections created for table string.
WARNING 4246: Only GLOBAL scope is supported for clearing string profiles
WARNING 4463: Projection string is not up to date
WARNING 4468: Projection <string> is not available for query processing. Execute the select start_refresh() function to copy data into this projection.
   The projection must have a sufficient number of buddy projections and all nodes must be up before starting a refresh
WARNING 4792: Storage option "string" will be ignored for external table "string"
WARNING 4871: System view string for tuning rule string is currently invalid
WARNING 4873: System view for tuning rule string does not exist
WARNING 4996: This request may deadlock the system. Please report the details to technical support
WARNING 5068: Total declared length of columns of one of the constraints exceeds the limit, truncation may happen
WARNING 5119: Udx code didn’t respond when Vertica tried to get function prototype for string in library string: string
WARNING 5448: View string is currently invalid
WARNING 5451: Violations of some of foreign key constraints may not be reported because of no privilege on the foreign tables
WARNING 5642: Projection string is not persistent or not up to date; it will not be copied
WARNING 5643: Projection string is prejoin projection; it will not be copied
WARNING 5717: No statistics has been exported. Either the DB is empty or you try to export an external table or you do not have access to the available objects
WARNING 5724: Segmentation clause contains a string - data loads may be slowed significantly
WARNING 5727: Sort clause contains a string - data loads may be slowed significantly
WARNING 5741: View string depends on other relations
WARNING 5819: Design could not be reset
WARNING 5821: Detected keys sharing the same case-insensitive key name
WARNING 5860: Due to the data isolation of temp tables with an on-commit-delete-rows policy, the compute_flextable_keys() and compute_flextable_keys_and_build_view() functions cannot access this table's data. The build_flextable_view() function can be used with a user-provided keys table to create a view, but involves a DDL commit which will delete the table's rows.

WARNING 5873: Failed to add table string of hcatalog schema string to catalog: string

WARNING 5875: Failed to alter table string of hcatalog schema string to catalog: string

WARNING 5880: Failed to describe table string in hcatalog database string: string

WARNING 5881: Failed to describe table string in schema string: HCatalog database string does not exist

WARNING 5884: Failed to list hcatalog tables of hcatalog schema string: string

WARNING 5886: Failed to mirror table string in schema string: string

WARNING 5909: Found and ignored keys with names longer than the maximum column-name length limit

WARNING 5912: HASH() arguments contain expressions that reference table columns

WARNING 5917: Ignored some keys since the total key count exceeds the view column limit

WARNING 5922: Insufficient privileges to alter table string

WARNING 5923: Insufficient privileges to drop table string

WARNING 5991: Projection basename "string" hint was ignored. "string" is used as the basename

WARNING 5993: Projection is irregularly segmented by column

WARNING 6053: The view creation involved a DDL commit which deleted the table's rows

WARNING 6256: Error while analyzing constraint(s) string on string

WARNING 6257: Error while creating LTT for analyzing constraints on string

WARNING 6356: No partitions have been swapped. Neither table has partitions fall in range

WARNING 6417: Swapped partitions are not immediately moved to a new storage location

WARNING 6515: Projection with name "string" already exists in schema "string" for anchor table "string.string"

WARNING 6565: Inequal query trees produced the same hash code query1:

```
string
query2:
string
```

WARNING 6594: Unknown RangeTblEntry: value

WARNING 6608: string is an invalid argument for hint string

WARNING 6765: Error parsing distrib value string in Distrib hint's arguments. The whole hint will be ignored

WARNING 6773: Failed to compare plans for query 'string'

WARNING 6797: Hint string can accept at most one argument, the hint with value arguments is ignored

WARNING 6798: Hint string can be specified only once for each join, if multiple instances listed only the first is considered

WARNING 6799: Hint string is not feasible and will be ignored

WARNING 6801: Hint string requires exactly two arguments, the hint with value arguments is ignored

WARNING 6813: Inherited privileges are globally disabled; schema parameter is set but has no effect

WARNING 6814: Inherited privileges are globally disabled; table parameter is set but has no effect

WARNING 6815: Inherited privileges are globally disabled; view parameter is set but has no effect

WARNING 6818: Input operations specified for Hint string is not feasible and will be ignored

WARNING 6922: Projection name was changed to string because it conflicts with the basename of the table string

WARNING 6990: Text index has invalid tokenizer
WARNING 6991: The active saved plan for this query has incompatible output column set. Continuing with the original query
WARNING 7000: The projection 'string' was used to enforce the enabled key constraint 'string', and may be regenerated to validate a DML statement on the base table
WARNING 7020: Unable to find the following query in the export file: 'string'
WARNING 7030: Unexpected group clause found during query comparison
WARNING 7031: Unexpected group clause found during query hashing
WARNING 7033: Unexpected sort clause found during query comparison
WARNING 7036: Unknown Expr:value during query comparison
WARNING 7037: Unknown Expr:value found
WARNING 7038: Unknown Expr:value found during query hashing
WARNING 7039: Unknown Inequality OpId:value found during query hashing
WARNING 7040: Unknown Logic OpExpr:value found during query comparison
WARNING 7041: Unknown Logic OpExpr:value found during query hashing
WARNING 7044: Unknown OpExpr:value found during query comparison
WARNING 7045: Unknown OpExpr:value found during query hashing
WARNING 7046: Unknown RangeTblEntry:value found during query comparison
WARNING 7047: Unknown RangeTblEntry:value found during query hashing
WARNING 7048: Unknown Sublink:value found during query comparison
WARNING 7049: Unknown Sublink:value found during query hashing
WARNING 7052: Unsupported JoinExpr Type found during query comparison
WARNING 7053: Unsupported JoinExpr Type found during query hashing
WARNING 7070: View "string" will include privileges from schema "string"
WARNING 7074: You appear to be using an old version of HDFS that may have stability problems under high load. See the Vertica documentation for supported HDFS versions
WARNING 7186: Inherited privileges are globally disabled. Privileges will be materialized and visibility of table string may change
WARNING 7264: Updates on the target table with a prejoin projection might ignore PK constraints and cause data integrity violations
WARNING 7265: Inherited privileges are globally disabled. Privileges will be materialized and visibility of view string may change
WARNING 7709: Could not drop table 'string'. Please remove manually if necessary. Detail: string
WARNING 7752: Cannot make a local query plan for "try local" hint, please check hash function, segment quantity, segment boundary, etc
WARNING 7793: The cache will be set to 1 instead
WARNING 7942: Unknown expression found during hashing: value
WARNING 7994: Not converged before max_iterations [value]
WARNING 8012: The parameter [string] is not defined for this function
WARNING 8086: Encoding option specified for attribute (string) of external table (string) will be ignored
WARNING 8275: The following shards missed the string due to missing primary subscriber (probably down): string
WARNING 8343: Not converged before max_iterations value
WARNING 8413: Lambda provided without regularization type: default for regularization is [string]; lambda will have no effect

WARNING 8414: max_depth is set to value while max_breadth to value. This means the size of trees may become limited by string first

WARNING 8419: Parameters [string] are not supported for optimizer [string], only for [string]

WARNING 8506: CGD optimizer could not invert covariance matrix which is required for calculating statistics. Coefficients are not affected

WARNING 8526: Regularization type [string] may cause optimizer [string] to not converge

WARNING 8531: string Directory for errors files was not created. Unable to write errors for this instance of COPY command

WARNING 8547: Not converged before maximum inner iterations value

WARNING 8557: Feature string not applicable in Eon mode

WARNING 8580: Failed to convert internal form to readable form: string

WARNING 8581: Failed to convert to internal form: string

WARNING 8582: Failed to describe hcatalog tables of hcatalog schema string: string

WARNING 8583: Given lambda value [value] will result in all zero coefficients; use a value lower than lambda max for this dataset [value]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Warning Messages Associated with SQLSTATE 01004

This topic lists the warnings associated with the SQLSTATE 01004.

SQLSTATE 01004 Description

ERRCODE_WARNING_STRING_DATA_RIGHT_TRUNCATION

Warning messages associated with this SQLState

WARNING 7166: client_label exceeded maximum length; truncated to 255 characters

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Warning Messages Associated with SQLSTATE 01006

This topic lists the warnings associated with the SQLSTATE 01006.

SQLSTATE 01006 Description

ERRCODE_WARNING_PRIVILEGE_NOT_REVOKED

Warning messages associated with this SQLState

WARNING 4925: The string "string" cannot be string string "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Warning Messages Associated with SQLSTATE 01007

This topic lists the warnings associated with the SQLSTATE 01007.

SQLSTATE 01007 Description

ERRCODE_WARNING_PRIVILEGE_NOT_GRANTED

Warning messages associated with this SQLState

WARNING 5682: USAGE privilege on schema "string" also needs to be granted to "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Warning Messages Associated with SQLSTATE 01V01

This topic lists the warnings associated with the SQLSTATE 01V01.

SQLSTATE 01V01 Description

ERRCODE_WARNING_DEPRECATED_FEATURE

Warning messages associated with this SQLState

- WARNING 2693: Configuration parameter string has been deprecated; setting it has no effect
- WARNING 4736: set_local_segment_threshold has been deprecated; setting it has no effect
- WARNING 7399: Created prejoin projection 'string'. Prejoin projections have been deprecated and will be removed in a future version
- WARNING 7585: Using prejoin projection 'string'. Prejoin projections have been deprecated and will be removed in a future version
- WARNING 8080: Creating projection "string" with explicit range segmentation. Range segmentation for projections has been deprecated and will be removed in a future version
- WARNING 8142: Query uses projection "string" which is explicitly range segmented. Range segmented projections have been deprecated and will be removed in a future version
- WARNING 8461: drop_partition has been deprecated and will be removed in a future version
- WARNING 8462: merge_partitions has been deprecated and will be removed in a future version
- WARNING 8463: merge_projection_partitions has been deprecated and will be removed in a future version

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 08000

This topic lists the errors associated with the SQLSTATE 08000.
SQLSTATE 08000 Description

ERRCODE_CONNECTION_EXCEPTION

Error messages associated with this SQLState

ERROR 2029: string from stdin failed: string
ERROR 2708: Connection to database [string] is invalid
ERROR 2896: Could not receive data from server: string
ERROR 2908: Could not send data to server: string
ERROR 3276: Error while waiting on socket. value
ERROR 4342: Password encryption failed
ERROR 5197: Unknown authentication method (value) requested by server
ERROR 7662: MD5 password hash requested when MD5 is not allowed

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 08001

This topic lists the errors associated with the SQLSTATE 08001.

SQLSTATE 08001 Description

ERRCODE_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION

Error messages associated with this SQLState

ERROR 2322: Cancel() -- connect() failed:
ERROR 2324: Cancel() -- socket() failed:
ERROR 2823: Could not connect to server [string]: string
   Is the server running and accepting
   TCP/IP connections on port string?
ERROR 2824: Could not connect to server: string
Is the server running on host [string] and accepting TCP/IP connections on port string?
ERROR 2839: Could not create socket: string
ERROR 2865: Could not get client address from socket: string
ERROR 2869: Could not get socket error status: string
ERROR 2912: Could not set socket to close-on-exec mode: string
ERROR 2913: Could not set socket to non-blocking mode: string
ERROR 2914: Could not set socket to TCP no delay mode: string
ERROR 7801: Could not translate host name "string" to address using family "string": string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 08003

This topic lists the errors associated with the SQLSTATE 08003.

SQLSTATE 08003 Description

ERRCODE_CONNECTION_DOES_NOT_EXIST

Error messages associated with this SQLState

ERROR 4717: Server closed the connection unexpectedly
This probably means the server terminated abnormally before or while processing the request

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 08006

This topic lists the errors associated with the SQLSTATE 08006.
SQLSTATE 08006 Description

ERRCODE_CONNECTION_FAILURE

Error messages associated with this SQLState

ERROR 2323: Cancel() -- send() failed: string
ERROR 2606: Client failed when looking for pending signals
ERROR 4539: Received no response from stringstring
ERROR 8466: Received no response from (string), operation (string)

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 08V01

This topic lists the errors associated with the SQLSTATE 08V01.

SQLSTATE 08V01 Description

ERRCODE_PROTOCOL_VIOLATION

Error messages associated with this SQLState

ERROR 2055: string Unexpected message type string reading from stdin
ERROR 2257: Bind message has value parameter formats but value parameters
ERROR 2258: Bind message has value result formats but query has value columns
ERROR 3334: Expected a RowDescription Message
ERROR 3335: Expected a SendExport Message
ERROR 3575: Insufficient data left in message
ERROR 3631: Invalid CLOSE message subtype value
ERROR 3651: Invalid DESCRIBE message subtype value
ERROR 3699: Invalid message format
ERROR 3701: Invalid message type
ERROR 3702: Invalid message type value
ERROR 3755: Invalid string in message
ERROR 3887: Lost synchronization with server: length value
ERROR 4074: No data left in message
ERROR 4718: Server did not identify with a pid & key
ERROR 5181: Unexpected message type 0xhex value
ERROR 5208: Unknown message from server
ERROR 5872: Expected to flush an end-of-batch client message but received a message of type value. Attempting to recover...
ERROR 6863: MARS operation not supported for your client version. Parameter not changed

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 09000

This topic lists the errors associated with the SQLSTATE 09000.

SQLSTATE 09000 Description

ERRCODE_TRIGGERED_ACTION_EXCEPTION

Error messages associated with this SQLState

ERROR 7353: A problem during the execution of writeModel.
   Detail: string
ERROR 7356: A problem occurred during the execution of a BFGS iteration.
   Detail: string
ERROR 7358: A problem occurred during the execution of a MinMax iteration.
   Detail: string
ERROR 7359: A problem occurred during the execution of a Newton iteration.
   Detail: string
ERROR 7360: A problem occurred during the execution of a robust_zscore iteration.
   Detail: string
ERROR 7361: A problem occurred during the execution of a zscore iteration.
   Detail: string
ERROR 7362: A problem occurred during the execution of an iteration.
   Detail: string
ERROR 7373: Cannot check if all columns are numeric.
  Detail: string

ERROR 7377: Cannot compute input column list.
  Detail: string

ERROR 7378: Cannot find the current user.
  Detail: string

ERROR 7383: Cannot make inf clause.
  Detail: string

ERROR 7384: Cannot make null clause.
  Detail: string

ERROR 7395: Could not compute evaluation metrics.
  Detail: string

ERROR 7396: Could not create view 'string'.
  Detail: string

ERROR 7497: Problem in balance.
  Detail: string

ERROR 7498: Problem in balance: Unknown exception

ERROR 7503: Problem in detect_outliers.
  Detail: string

ERROR 7504: Problem in detect_outliers: Unknown exception

ERROR 7505: Problem in initializing centers table.
  Detail: string

ERROR 7506: Problem in kmeans.
  Detail: string

ERROR 7507: Problem in kmeans: Unknown exception

ERROR 7516: Problem in normalize.
  Detail: string

ERROR 7517: Problem in normalize: Unknown exception

ERROR 7519: Problem in reading the information of the model.
  Detail: string

ERROR 7523: Problem in summarize_model.
  Detail: string

ERROR 7524: Problem in writing initial model to DFS.
  Detail: string

ERROR 7593: Could not randomly pick value distinct centers after trying value times

ERROR 7611: Problem in summarize_model: Unknown Exception

ERROR 7626: A problem during the execution of computing the number of detected outliers.
  Detail: string

ERROR 7731: Problem in calculating alpha for Linear Regression.
  Detail: string

ERROR 7732: Problem in calculating Hessian matrix for Linear Regression.
  Detail: string

ERROR 7733: Problem in strongWolfeLineSearch.
  Detail: string

ERROR 7734: Problem in zoom.
  Detail: string

ERROR 7761: A problem during the execution of computing the mad.
  Detail: string
ERROR 7762: A problem during the execution of computing the median.
   Detail: string
ERROR 7763: A problem occurred during the execution of a mad computation.
   Detail: string
ERROR 7764: A problem occurred during the execution of median computation.
   Detail: string
ERROR 7769: Kmeans++ exceeded max number of retries for choosing initial center
ERROR 7784: Could not remove blob named string.
   Detail: Unexpected exception
ERROR 7785: Could not remove the blob named string.
   Detail: string
ERROR 7805: No input columns provided
ERROR 7826: Numeric overflow occurred during execution of kmeans++
ERROR 7827: Numeric overflow occurred when computing total sum of squares
ERROR 7828: Numeric overflow occurred when computing total within-cluster sum of squares
ERROR 7829: Numeric overflow occurred when computing within-cluster sum of squares
ERROR 7844: Input table string is empty
ERROR 7845: No input columns left after excluding
ERROR 7851: No rows remain after filtering rows with null and infinity values
ERROR 7858: Only found value non-empty clusters. You may want to try again or use a better set of initial centers; or there may be fewer than k distinct datapoints in the table
ERROR 7947: A problem occurred during table renaming.
   Detail: string
ERROR 7949: A problem occurred during the execution of mode computation.
   Detail: string
ERROR 7952: A problem occurred during the execution of computing mode for the first MiniBatch.
   Detail: string
ERROR 7953: A problem occurred during the execution of computing mode.
   Detail: string
ERROR 7955: A problem occurred during the execution of MiniBatch Table Merging.
   Detail: string
ERROR 7961: Problem in impute.
   Detail: string
ERROR 7962: Problem in impute: Unknown exception
ERROR 7975: A problem occurred during training of SVM model.
   Detail: string
ERROR 7988: Input table is empty
ERROR 7992: No rows remain after filtering rows containing NULL, NaN or INF
ERROR 8000: Problem in normalize_fit.
   Detail: string
ERROR 8001: Problem in normalize_fit: Unknown exception
ERROR 8014: There was a problem reading the model summary.
   Detail: string
ERROR 8020: Unexpected empty result from query
ERROR 8035: A problem occurred during the execution of DROP TABLE impute_201701_80425147_p1.
    Detail: string
ERROR 8036: A problem occurred during the execution of DROP TABLE impute_201701_80425147_p1,impute_201701_80425147_p2.
    Detail: string
ERROR 8045: A blob container cannot be initialized more than once
ERROR 8046: A problem during the execution of computing the max weight.
    Detail: string
ERROR 8047: A problem during the execution of sse_linear_reg
ERROR 8048: A problem in the string summary.
    Detail: string
ERROR 8049: A problem in the summary function.
    Detail: string
ERROR 8055: A problem occurred during the execution of computing weight table.
    Detail: string
ERROR 8056: A problem occurred during the execution of saving weight table to vectors.
    Detail: string
ERROR 8078: Could not link model string to catalog object.
    Details: string
ERROR 8102: Invalid value [string] for parameter [string]. Valid values are [string]
ERROR 8113: Model name cannot be empty
ERROR 8134: Problem in factor function.
    Detail: string
ERROR 8241: A problem occurred during the execution of a mode imputation with partition by.
    Detail: string
ERROR 8242: A problem occurred during the execution of a mode imputation without partition by.
    Detail: string
ERROR 8243: A problem occurred during the execution of a mean imputation with partition by.
    Detail: string
ERROR 8244: A problem occurred during the execution of a mean imputation.
    Detail: string
ERROR 8291: A problem occurred during the execution of coordinate descent covariance.
    Detail: string
ERROR 8293: A problem occurred during the execution of offsets query (averages of all columns).
    Detail: string
ERROR 8301: Parameter [string] cannot be redefined
ERROR 8306: Problem in creating model [string].
    Detail: string
ERROR 8307: Problem in creating model [string]: Unknown exception
ERROR 8308: Problem in cross_validation.
    Detail: string
ERROR 8309: Problem in cross_validation: Unknown exception
ERROR 8315: Type [string] of parameter [string] is not supported
ERROR 8318: Empty string cannot be passed into extra_levels parameter
ERROR 8319: Invalid JSON for extra_levels: [string]
ERROR 8320: Problem in one_hot_encoder_fit.
  Detail: string
ERROR 8321: Problem in one_hot_encoder_fit: Unknown exception
ERROR 8344: Parameter [string] was provided and cannot be included again in cv_hyperparams
ERROR 8346: The total number of models to be trained is very large [value]. It could take very long time to finish
ERROR 8369: Could not train any tree. Increase sampling_size
ERROR 8399: A problem in model summary.
  Detail: string
ERROR 8405: Could not create catalog object for model [string]
  Detail: string
ERROR 8406: Could not create catalog object for temporary model
  Detail: string
ERROR 8407: Could not link temporary model to catalog object.
  Details: string
ERROR 8420: Problem in creating temporary model.
  Detail: string
ERROR 8421: Problem in creating temporary model: Unknown exception
ERROR 8499: A problem occurred during training of a Naive Bayes model.
  Detail: string
ERROR 8507: Could not upgrade model to latest version, during import. Please run upgrade_model().
  DETAIL: string
ERROR 8508: Could not upgrade model to latest version, during import. Please run upgrade_model(). Unknown Exception
ERROR 8512: Function summarize_model is deprecated. Use get_model_summary to summarize models
ERROR 8523: One of the folds does not have enough data to train. Please try to reduce the number of folds
ERROR 8529: Unsupported model format version for upgrade
ERROR 8565: Relation [string] has no valid rows
ERROR 8566: Relation [string] is empty
ERROR 8571: Cannot upgrade an incomplete model: [value]. Skipping this model
ERROR 8572: Error while initializing model upgrade task.
  Details: string
ERROR 8573: Error while performing upgrade for model: string.
  Details: Model file: string does not exists
ERROR 8574: Error while performing upgrade for model[value].
  Details: Model does not exists
ERROR 8575: Error while performing upgrade for model[value].
  Details: string
ERROR 8576: Error while performing upgrade.
  Details: string
ERROR 8577: Error while upgrade models.
  DETAILS: string
ERROR 8590: Schema value not found
Note: The **Vertica User Community** contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

## Error Messages Associated with SQLSTATE 0A000

This topic lists the errors associated with the SQLSTATE 0A000.

### SQLSTATE 0A000 Description

**ERRCODE_FEATURE_NOT_SUPPORTED**

### Error messages associated with this SQLState

- ERROR 2009: *string* can not be used in function *string*
- ERROR 2013: *string* clause is not supported for expressions
- ERROR 2014: *string* Concatenated GZIP/BZIP is not supported with NATIVE/NATIVE VARCHAR formats
- ERROR 2036: *string* is not a legal time unit
- ERROR 2058: *string VIEW* is not supported
- ERROR 2009: A correlated column in a subquery expression is not supported
- ERROR 2114: ADD COLUMN over temporary tables is not supported
- ERROR 2130: Aggregate function *string (value)* is not supported
- ERROR 2133: Aggregate function calls cannot contain subqueries
- ERROR 2138: Aggregate functions can only be called on columns of a table
- ERROR 2161: ALL subquery with a correlated expression is not supported
- ERROR 2165: ALTER COLUMN TYPE over temporary tables is not supported
- ERROR 2166: ALTER TABLE does not support ADD COLUMN with other clauses
- ERROR 2167: ALTER TABLE does not support ALTER COLUMN TYPE with other clauses
- ERROR 2168: ALTER TABLE does not support DROP COLUMN with other clauses
- ERROR 2169: ALTER TABLE does not support SET SCHEMA with other clauses
- ERROR 2178: An expression containing a correlated subquery with aggregate function is not supported
- ERROR 2183: Analytic functions are not allowed in an ORDER BY on a UNION/INTERSECT/EXCEPT
- ERROR 2184: Analytic functions are not supported in the ORDER BY of an analytic function OVER clause
- ERROR 2190: Analytics query with having clause expression that involves aggregates and subquery is not supported
- ERROR 2192: ANALYZE_CONSTRAINTS is currently not supported in non-default locales
- ERROR 2208: Another Design/Deployment is in progress
ERROR 2210: ANTI join with segmented inner not supported
ERROR 2220: Argument string must not contain subqueries
ERROR 2226: Argument to seeded random must be a constant
ERROR 2233: Array References are not supported
ERROR 2235: ArrayExpr is not supported
ERROR 2329: Cannot accept a value of type any
ERROR 2330: Cannot accept a value of type anyarray
ERROR 2331: Cannot accept a value of type anyelement
ERROR 2332: Cannot accept a value of type internal
ERROR 2333: Cannot accept a value of type language_handler
ERROR 2334: Cannot accept a value of type opaque
ERROR 2335: Cannot accept a value of type trigger
ERROR 2340: Cannot add IDENTITY/AUTO-INCREMENT columns
ERROR 2350: Cannot alter type of column "string" since it is referenced in the constraint "string"
ERROR 2351: Cannot alter type of column "string" since it is referenced in the default expression of column "string"
ERROR 2352: Cannot alter type of column "string" since it is referenced in the partition expression
ERROR 2353: Cannot alter type of column "string" since it is referenced in the segmentation expression of projection "string"
ERROR 2354: Cannot alter type of column with a default expression
ERROR 2360: Cannot assign to system column "string"
ERROR 2363: Cannot broadcast non-subquery outer input to a join
ERROR 2368: Cannot change owner of temporary table
ERROR 2377: Cannot convert column "string" from "string" to type "string"
ERROR 2392: Cannot delete from a view
ERROR 2399: Cannot display a value of type any
ERROR 2400: Cannot display a value of type anyelement
ERROR 2401: Cannot display a value of type internal
ERROR 2402: Cannot display a value of type language_handler
ERROR 2403: Cannot display a value of type opaque
ERROR 2404: Cannot display a value of type trigger
ERROR 2407: Cannot drop a table column when a node is down
ERROR 2411: Cannot drop column "string" since it is referenced in the partition expression
ERROR 2412: Cannot drop column "string" since it is referenced in the primary key constraint
ERROR 2425: Cannot export virtual string string
ERROR 2443: Cannot insert into a view
ERROR 2458: Cannot mergeout uncommitted data in the presence of savepoints
ERROR 2461: Cannot moveout uncommitted data in the presence of savepoints
ERROR 2503: Cannot set a subfield to DEFAULT
ERROR 2504: Cannot set an array element to DEFAULT
ERROR 2532: Cannot update a view
ERROR 2533: Cannot Update/Merge tables involved in prejoins with Limit clause on unsorted output
ERROR 2549: Cannot use DISTINCT with user-defined transform functions
ERROR 2552: Cannot use meta function or non-deterministic function in PARTITION BY expression
ERROR 2556: Cannot use SAVEPOINT with uncommitted tuple mover enabled
ERROR 2558: Cannot use subquery in EXECUTE parameter
ERROR 2559: Cannot use subquery in expressions within COPY
ERROR 2560: Cannot use subquery in PARTITION BY expression
ERROR 2561: Cannot use subquery in SEGMENTED BY expression
ERROR 2562: Cannot use Vertica’s built-in file source and a UDSource in the same query
ERROR 2569: Catalog object string does not exist
ERROR 2602: Clause "NO PROJECTION" conflicts with the column list
ERROR 2683: Clause "NO PROJECTION" is supported only on temporary tables
ERROR 2688: CoerceToDomainValue is not supported
ERROR 2698: ConvertRowtypeExpr is not supported
ERROR 2767: Command string is not supported
ERROR 2768: COLUMN not supported for system objects
ERROR 2769: COMMENT not supported for this object type
ERROR 2770: Conditional UNION/INTERSECT/EXCEPT statements are not implemented
ERROR 2771: Conflicting or redundant column options
ERROR 2772: Correlated EXISTS/NOT EXISTS subquery containing having clause with aggregates is not supported
ERROR 2773: Correlated EXISTS/NOT EXISTS subquery with limit 0 is not supported
ERROR 2776: Correlated EXISTS/NOT EXISTS with User Defined Aggregate is not supported
ERROR 2777: Correlated expression in ON clause is not supported
ERROR 2778: Correlated expression in set operator subquery is not supported
ERROR 2779: Correlated expressions in SELECT list of subquery are not supported
ERROR 2780: Correlated subqueries cannot have more than one level
ERROR 2781: Correlated subqueries with analytics in the select list is not supported
ERROR 2782: Correlated subqueries with no group by and a non-strict expression containing an aggregate in the select list is not supported
ERROR 2783: Correlated subquery column in select/gby/oby not supported
ERROR 2784: Correlated subquery could not be flattened as a join
ERROR 2785: Correlated subquery could not get flattened, a correlated expression could not be treated as a join
ERROR 2786: Correlated subquery expression without aggregates and with limit is not supported
ERROR 2787: Correlated subquery expressions under OR is not supported
ERROR 2788: Correlated subquery in expression with operator <> is not supported
ERROR 2790: Correlated subquery with aggregate and limit 0 is not supported
ERROR 2792: Correlated subquery with aggregate function COUNT is not supported
ERROR 2793: Correlated subquery with distinct/group by is not supported
ERROR 2794: Correlated subquery with having clause expression that involves aggregates and subquery is not supported
ERROR 2795: Correlated subquery with NOT IN is not supported
ERROR 2796: Correlated subquery with outer joins and uncorrelated exists is not supported
ERROR 2797: Correlated subquery with User Defined Aggregate is not supported
ERROR 2854: Could not find array type for data type string
ERROR 2856: Could not find column string in table string
ERROR 2942: CREATE ASSERTION is not supported
ERROR 2943: CREATE FUNCTION / INOUT parameters are not supported
ERROR 2944: CREATE FUNCTION / OUT parameters are not supported
ERROR 2980: Data type not supported
ERROR 2981: Data type not supported (value)
ERROR 2983: Database "string" does not exist
ERROR 2987: Database references are not supported: "string.string.string"
ERROR 3026: Defining query must have a from clause
ERROR 3115: DistinctExpr not supported
ERROR 3116: Distrib overrides are too restrictive. Can not find completed Join Order
ERROR 3118: DML on projection/view is not supported
ERROR 3119: DML query with a predicate that could not be pushed below joins and does not refer solely to the target table is not supported
ERROR 3123: DROP ASSERTION is not supported
ERROR 3126: DROP COLUMN over temporary tables is not supported
ERROR 3132: DROP SEQUENCE does not support CASCADE
ERROR 3141: Dropping local and global objects in one statement is not supported
ERROR 3157: Dynamic load not supported
ERROR 3163: Embedded SQL involving local objects is not supported
ERROR 3174: ENCODED BY is supported in CREATE TABLE ... AS SELECT statement only
ERROR 3246: Error parsing distrib overrides (unexpected end of override); string
ERROR 3247: Error parsing distrib value; string
ERROR 3291: Event ANY_ROW is not supported
ERROR 3317: Executing when OPT:PLAN_ALL_NODES_ACTIVE option is set
ERROR 3343: Explicit JOIN clause contains a join predicate between relations previously joined
ERROR 3351: Expressions in COPY may not contain aggregate functions
ERROR 3352: Expressions in COPY may not contain analytic or Time Series Aggregate Functions
ERROR 3353: Expressions not supported in Times Series Aggregate Function
ERROR 3357: External tables only support files or a User Defined Source
ERROR 3403: FieldSelect is not supported
ERROR 3404: FieldStore is not supported
ERROR 3417: Final phase output size mismatch
ERROR 3420: First argument of date_part must be a constant string
ERROR 3434: For INSERT SELECT statement, replicated/broadcasted source data not supported
ERROR 3436: For SELECT DISTINCT, ORDER BY expressions must appear in the SELECT clause
ERROR 3451: Function string can’t be used as a case expression
ERROR 3452: Function string can’t be used in a boolean
ERROR 3453: Function string can’t be used in a WHEN clause
ERROR 3454: Function string can’t be used in as a segment expression
ERROR 3455: Function string can’t be used with an operator
ERROR 3488: Group By, Order By, Aggregates, Having & limits not allowed in update/delete
ERROR 3510: IGNORE NULLS argument must be a Boolean constant
ERROR 3553: INHERITS not supported
ERROR 3566: Input of anonymous composite types is not implemented
ERROR 3600: Interpolated predicates can accept arguments of the same type only
ERROR 3601: Interpolated predicates can be part of AND expressions only
ERROR 3613: Interval units "string" not supported
ERROR 3821: Joins with an interpolated predicate can have a conjunctive expression containing equality predicates. The equality predicates cannot have expressions or column references with different modifiers
ERROR 3822: Joins with an interpolated predicate cannot have expressions or column references with different modifiers in any of the expressions
ERROR 3857: Library built with unsupported version of Vertica SDK [Version: string, Revision: string]
ERROR 3876: Locale must be a constant
ERROR 3900: MATCH PARTIAL is not supported
ERROR 3972: Multi-column subquery expressions can only be used with the =, <=> and <> operators
ERROR 3973: Multi-column subquery type ALL can only be used with the = and <=> operators
ERROR 3974: Multi-column subquery type ANY can only be used with the =, <=> and <> operators
ERROR 4106: No single-source bulk loads have been executed in this session
ERROR 4147: Node issuing the query cannot be marked as down
ERROR 4160: Non-equality correlated subquery expression is not supported
ERROR 4170: Not a Star or Snow-Flake Query block
ERROR 4171: Not a Star or Snow-Flake Query block; dimension table not a star or snowflake
ERROR 4172: Not a Star or Snow-Flake Query block; no fact table found
ERROR 4173: Not a Star or Snow-Flake Query block; there are multiple fact tables
ERROR 4197: NULL value found in a column used by a subquery
ERROR 4228: ON COMMIT DROP not supported in CREATE TABLE
ERROR 4238: Only a temporary table projection can be pinned
ERROR 4248: Only inner joins are allowed in the projection defining query
ERROR 4256: Only relations and subqueries are allowed in the FROM clause
ERROR 4258: Only super user can call export_catalog with an output file name
ERROR 4259: Only super user can get the rebalance data script
ERROR 4263: Only superuser can drop system schema
ERROR 4264: Only superuser can rebalance data
ERROR 4265: Only superuser can rebalance data for replicated projections
ERROR 4266: Only superuser can rebalance data for segmented projections
ERROR 4280: Operator string (value) is not supported
ERROR 4281: Operator string is not supported for row expressions
ERROR 4298: ORDER BY on a UNION/INTERSECT/EXCEPT result must be on one of the result columns
ERROR 4299: ORDER mode not supported
ERROR 4306: OUTER join with broadcasted outer data not supported
ERROR 4307: OUTER or SEMI join - done through CROSS join and FILTER - with replicated outer and segmented inner not supported
ERROR 4308: OUTER relation in OUTER join is not the fact table nor a snowflake dimension table
ERROR 4309: Outer replicated/segmented input to a join cannot be resegmented
ERROR 4310: LEFTOUTER/SEMI/ANTI join with replicated/broadcasted outer data not supported
ERROR 4328: PARTITION AUTO can only be used with single-phase user defined transform functions
ERROR 4329: PARTITION AUTO cannot be used with pattern matching
ERROR 4331: PARTITION BY expression cannot return a tuple
ERROR 4332: PARTITION BY expression has an unknown type
ERROR 4333: PARTITION BY expression may not contain aggregate functions
ERROR 4335: Partitioning expression not supported for temporary tables
ERROR 4336: Partitioning not supported for temporary tables
ERROR 4352: Pattern "E" is not supported
ERROR 4375: PINNED clause conflicts with KSAFE setting
ERROR 4376: PINNED clause is not supported in CREATE TABLE statement
ERROR 4412: Prepared statements are currently unsupported
ERROR 4465: Projection string of local temporary table cannot be created under user schema string
ERROR 4471: Projection choices are too restrictive - cannot create correct join between tables
ERROR 4486: Projections are always created and persisted in the default Vertica locale. The current locale is string
ERROR 4584: RENAME COLUMN over temporary tables is not supported
ERROR 4586: replicate_catalog has been shut off
ERROR 4628: Row Expressions are not supported in this context
ERROR 4631: ROW syntax is not supported
ERROR 4644: Scalar array expression cannot contain column references or subqueries
ERROR 4645: Scalar array op string (value) is not supported
ERROR 4664: Segmentation clause can not have offset in CREATE TABLE statement
ERROR 4665: Segmentation clause with offset conflicts with KSAFE setting
ERROR 4666: Segmentation expression must have integer type
ERROR 4671: SELECT FOR UPDATE cannot be applied to a function
ERROR 4672: SELECT FOR UPDATE cannot be applied to a join
ERROR 4673: SELECT FOR UPDATE cannot be applied to NEW or OLD
ERROR 4674: SELECT FOR UPDATE is not allowed with EXTERNAL TABLES
ERROR 4675: SELECT FOR UPDATE is not allowed with libraries
ERROR 4676: SELECT FOR UPDATE is not allowed with sequences
ERROR 4677: SELECT FOR UPDATE is not allowed with UNION/INTERSECT/EXCEPT
ERROR 4678: SELECT FOR UPDATE is not allowed with views
ERROR 4680: Self joins in UPDATE statements are not allowed
ERROR 4703: Sequence cannot be moved between system schema and user schema
ERROR 4711: Sequence or IDENTITY/AUTO_INCREMENT column in merge query is not supported
ERROR 4714: Sequences are not allowed in default expressions of local temp tables
ERROR 4715: Sequences cannot be called in views
ERROR 4716: Sequences cannot be created under system schemas
ERROR 4728: Set Operator string ALL not supported
ERROR 4730: Set Operator queries without a FROM clause are not supported
ERROR 4733: SET SCHEMA over temporary tables is not supported
ERROR 4735: Set-valued function called in context that cannot accept a set
ERROR 4747: SetToDefault is not supported
ERROR 4786: Statement string is not supported
ERROR 4808: Subqueries are not supported as the left hand argument to another subquery
ERROR 4809: Subqueries are not supported in the ORDER BY of a timeseries OVER clause
ERROR 4810: Subqueries are not supported in the ORDER BY of an analytic function OVER clause
ERROR 4812: Subqueries are not supported in the PARTITION BY of an analytic function OVER clause
ERROR 4816: Subqueries in the ON clause are not supported
ERROR 4817: Subqueries in the SELECT or ORDER BY are not supported if the query has aggregates and the subquery is not part of the GROUP BY
ERROR 4818: Subqueries in the SELECT or ORDER BY are not supported if the subquery is not part of the GROUP BY
ERROR 4820: Subqueries in UPDATE/DELETE/MERGE is not supported
ERROR 4821: Subqueries not allowed in target of insert
ERROR 4822: Subqueries referring to no outer columns in HAVING clause when query has aggregates and no GROUP BY are not supported
ERROR 4824: Subquery aggregate expression that refers a correlated column is not supported
ERROR 4839: Subquery type ARRAY is not supported
ERROR 4842: Subquery without a from clause is not supported
ERROR 4850: Support for UPDATE/DELETE/MERGE is not enabled
ERROR 4854: SyncMarkers are not supported
ERROR 4865: System table string cannot be created under user schema string
ERROR 4869: System view "string" cannot be dropped
ERROR 4870: System view string cannot be created under user schema string
ERROR 4884: Table string cannot be created under system schema string
ERROR 4897: Table cannot be moved between system schema and user schema
ERROR 4910: Table revalidation error
ERROR 4918: Temporary Sequences are not supported
ERROR 4933: The argument types in a subquery expression in the where/hAVING clause do not match
ERROR 4938: The constant value following the LIMIT clause cannot be negative
ERROR 4939: The constant value following the OFFSET clause cannot be negative
ERROR 4948: The fourth input argument of TIME_SLICE must be START or END
ERROR 4960: The ORDER BY ... USING clause is not supported
ERROR 4966: The second parameter of export_catalog is invalid: string
ERROR 4968: The slice length parameter of TIME_SLICE must be a positive integer
ERROR 5005: Time Series Aggregate Function with interpolation scheme LINEAR may only have an INTEGER or FLOAT type as its first argument
ERROR 5016: Time units "string" not supported
ERROR 5023: Timeseries output functions are not supported in the ORDER BY of a timeseries OVER clause
ERROR 5028: Timestamp units "string" not supported
ERROR 5110: Type string (value) is not supported
ERROR 5159: Uncorrelated EXISTS subqueries are not supported when the query has both HAVING clause subqueries involving aggregates and when the query has either OUTER JOINS or NOT IN subqueries
ERROR 5160: Uncorrelated EXISTS subqueries in HAVING clause when query has aggregates and no GROUP BY are not supported
ERROR 5195: UNIQUE predicate is not supported
ERROR 5262: Unsafe use of string constant with Unicode escapes
ERROR 5264: Unsupported access to session-scoped (LOCAL) object
ERROR 5270: Unsupported COPY command clause
ERROR 5275: Unsupported Join in From clause
ERROR 5276: Unsupported Join in From clause: FULL OUTER JOINS not supported
ERROR 5278: Unsupported join of two non-alike segmented projections
ERROR 5280: Unsupported mix of Joins
ERROR 5284: Unsupported query syntax
ERROR 5289: Unsupported subquery expression
ERROR 5291: Unsupported use of aggregates
ERROR 5292: Unsupported use of cursors
ERROR 5293: Unsupported use of DISTINCT clause
ERROR 5294: Unsupported use of FROM clause
ERROR 5295: Unsupported use of GROUP BY or DISTINCT clause
ERROR 5296: Unsupported use of HAVING clause
ERROR 5297: Unsupported use of LIMIT/OFFSET clause
ERROR 5298: Unsupported use of ORDER BY clause
ERROR 5299: Unsupported use of outer joins
ERROR 5300: Unsupported use of query/subquery without FROM clause
ERROR 5301: Unsupported use of sub-queries
ERROR 5302: Unsupported use of target relation
ERROR 5303: Unsupported use of UDF in WHERE clause
ERROR 5304: Unsupported use of UNION/INTERSECT/EXCEPT
ERROR 5313: Update is disallowed on Primary/Foreign Keys columns. Use Delete followed by Insert instead
ERROR 5314: UPDATE may not refer to tables in prejoin projections
ERROR 5366: User defined aggregate cannot be used in query with other distinct aggregates
ERROR 5388: User has insufficient privilege on string string
ERROR 5392: User must have the DBDUSER role to run the database designer
ERROR 5396: User projection string cannot be created under system schema string
ERROR 5402: User-defined transform functions are not supported in the ORDER BY clause
ERROR 5407: VALINDEX column must be the first column in ORDER BY list
ERROR 5426: Vertica currently allows a maximum of value physical storage containers per projection
ERROR 5427: Vertica does not support GRANT / REVOKE ON LANGUAGE
ERROR 5428: Vertica does not support GRANT / REVOKE ON TABLESPACE
ERROR 5447: View string cannot be created under system schema string
ERROR 5456: Volatile functions may not be used in fillers when other computed columns refer to them
ERROR 5465: Window frame exclusion is not supported
ERROR 5530: Audit of external tables is not supported
ERROR 5537: Cannot alter user-defined type "string" of column "string"
ERROR 5550: COPY from UDSOURCE does not support rejected row numbers with exceptions or rejected data options
ERROR 5551: COPY LOCAL cannot process more than ONE NATIVE or NATIVE VARCHAR file at a time
ERROR 5562: Creating temp tables by LIKE clause is not supported
ERROR 5595: Invalid argument type string in function string
ERROR 5607: Language of replacement library [string] must match language of existing library [string]
ERROR 5681: Unsupported base type string for User-defined type string
ERROR 5698: Cannot export statistics for the specified object
ERROR 5725: Size specification not supported for User Defined Type string
ERROR 5731: The second parameter must be a table/projection/column name
ERROR 5758: Can not drop Filesystem proc string
ERROR 5759: Can not drop library "string": referenced by storage locations
ERROR 5763: Can’t create a managed external table with non-file sources
ERROR 5764: Cannot alter the data type of a table column when a node is down
ERROR 5781: Cannot use meta function or non-deterministic function in SEGMENTED BY expression
ERROR 5859: Due to the data isolation of temp tables with an on-commit-delete-rows policy, the compute_flexibletable_keys() and compute_flexibletable_keys_and_build_view() functions cannot access this table's data
ERROR 5864: Error parsing table (invalid table): string
ERROR 5914: HCatalog schema string not permitted in search path
ERROR 5990: Projection string cannot be created under hcatalog schema string
ERROR 5992: Projection cannot be created for HCatalog table string
ERROR 6005: Remote table string.string found in design query
ERROR 6019: Sequence string cannot be created under hcatalog schema string
ERROR 6020: Sequence string cannot be moved between system schema and hcatalog schema string
ERROR 6023: Setting the CPU affinity of the built-in pool "string" is not supported
ERROR 6038: Table string cannot be created under hcatalog schema string
ERROR 6039: Table string cannot be moved under hcatalog schema string
ERROR 6092: Unsupported access to flex table: No string support
ERROR 6108: View string cannot be created under hcatalog schema string
ERROR 6141: ALTER COLUMN TYPE over tables having aggregate projections is not supported
ERROR 6142: Alter partition not supported for tables with aggregate projections
ERROR 6145: Analytic functions are not allowed in projections
ERROR 6168: Cannot drop column of a table with aggregate projections
ERROR 6173: Cannot open file string for debugging ExprHashCode
ERROR 6174: Cannot open/write/close file "string" for debugging ExprHashCode
ERROR 6200: Command CREATE INDEX is not supported
ERROR 6233: DISTINCT Aggregates are not allowed in projections
ERROR 6241: DROP COLUMN over tables having aggregate projections is not supported
ERROR 6244: EE option DISABLE_AUTOPARTITION may not be used in conjunction with aggregate projections, or projections with expressions
ERROR 6265: Expressions in the projection SELECT list may not be repeated
ERROR 6266: Expressions on aggregates are not allowed in aggregate projections
ERROR 6322: LIMIT may not be used in projection definitions unless the OVER clause is supplied
ERROR 6323: LIMIT or OVER(...) clause may not be used in projection definitions
ERROR 6332: MERGE is not supported on any target table having projection with expression
ERROR 6380: PARTITION BEST cannot be used with pattern matching
ERROR 6382: PARTITION NODES cannot be used with pattern matching
ERROR 6400: Setting the cascade to pool of the built-in pool "string" is not supported
ERROR 6401: Setting the cascade to pool to the built-in pool "string" is not supported
ERROR 6427: The LIMIT clause does not support OFFSET ... and OVER(...) clauses simultaneously
ERROR 6455: Unsupported access to table with projection expressions or aggregates
ERROR 6459: UPDATE/DELETE/MERGE a dim table where fact table has aggregate projections is not supported
ERROR 6460: UPDATE/DELETE/MERGE a table with aggregate projections is not supported
ERROR 6464: Use of LIMIT with the OVER(...) clause and DISTINCT is not supported within the same SELECT block
ERROR 6465: Use of LIMIT with the OVER(...) clause and ORDER BY is not supported within the same SELECT block
ERROR 6467: User defined aggregate cannot be used in query with MLAs
ERROR 6476: Cannot directly modify index table constraint
ERROR 6477: Cannot modify index table columns directly
ERROR 6501: Access policy cannot be created on table "string" since it has an aggregate projection
ERROR 6502: Access policy cannot be created on table "string" since it has a pre-join projection
ERROR 6503: Access policy cannot be created on table "string" since it has a projection with expressions
ERROR 6505: Cannot create Aggregate projection on tables with access policy
ERROR 6506: Cannot create prejoin projection on tables with access policy
ERROR 6507: Cannot create projection with expression on tables with access policy
ERROR 6517: Access policy is requested from unsupported object: "string"
ERROR 6522: Cannot alter column width on variable length columns with BLOCKDICT COMP encoding
ERROR 6525: Direct query to projection "string" are not supported: "string"
ERROR 6530: Invalid statistics file. Total number of bounds specified: 'value' do not match the buckets value: 'value' for column 'string'. Buckets value specified should exactly be same as total number of bounds
ERROR 6531: Invalid table name: 'string'. Make sure the schema/table exists in database
ERROR 6532: Invalid value 'value' for attribute 'string' under bound 'string', column 'string'. Please use a positive value.
ERROR 6533: Invalid value 'value' for attribute 'string' under column 'string'. Please use a positive value.
ERROR 6538: Unable to string: "string"
ERROR 6644: Arguments to Transform Functions in Live Aggregate Projections cannot be constants
ERROR 6670: Cannot alter a table constraint when a node is down
ERROR 6685: Cannot specify anything other than expressions on table columns in the SELECT list
ERROR 6690: Cannot use multi phase UDTs with live aggregate projections
ERROR 6742: drop_location for non-retired DATA locations is not supported
ERROR 6785: For NULLAWARE ANTI JOIN, columns of left-hand-side relation must not appear on the right hand side of ON clause predicates
ERROR 6786: For NULLAWARE ANTI JOIN, columns of right-hand-side relation must appear on the right hand side of ON clause predicates
ERROR 6787: For NULLAWARE ANTI JOIN, columns of right-hand-side relation must not appear on the left hand side of ON clause predicates
ERROR 6817: Input filename cannot be NULL
ERROR 6822: Inputs to batch transform function have to be in the same order as the outputs of the prepass transform function
ERROR 6823: Inputs to batch Transform function may not be expressions
ERROR 6852: Live Aggregate Projection "string" will be created for "string". Data in "string" will be neither updated nor deleted
ERROR 6862: MARS operation not supported for this client type. Parameter not changed
ERROR 6865: Materialized WITH queries are not supported with directed queries
ERROR 6867: Meta-functions cannot be used in directed queries
ERROR 6882: Null value in ON clause is not supported for Nullaware anti join
ERROR 6888: Only equality join is supported in ON clause of Nullaware anti join
ERROR 6919: Order BY clauses are not allowed in aggregate projection definitions
ERROR 6984: Output filename cannot be NULL
ERROR 6974: System/Virtual tables, projections or views not supported for optimizer-generated annotated queries
ERROR 6983: Tables or projections with access policies not supported for optimizer-generated annotated queries
ERROR 6992: The batch and prepass transform functions must be partitioned by the same values
ERROR 6996: The DB admin has disallowed using the MARS feature
ERROR 7054: Unsupported operation in ON clause of Nullaware anti join
ERROR 7103: Java user defined functions are not supported in aggregate projections
ERROR 7113: Queries with syntactic hints are not supported with directed queries
ERROR 7123: Statement including date/time literals based on current date/time are not supported for
           directed queries
ERROR 7124: Statement including hints are not supported for directed queries
ERROR 7133: UNI Join with non-subquery inner not supported
ERROR 7169: Constant true/false predicate with ignore constant hint is not supported
ERROR 7188: Object type value not supported for access policies
ERROR 7201: The constant value following the LIMIT clause cannot be both zero and ignoreconst
ERROR 7239: Computing flex table keys with non-binary collation locales is not supported
ERROR 7288: Transaction isolation level not supported. Parameter not changed
ERROR 7294: string expressions must not return a set
ERROR 7307: Cannot use aggregate functions in string expressions
ERROR 7308: Cannot use analytic or time series aggregate functions in string expressions
ERROR 7318: Default expressions with subqueries are not supported in a MERGE statement
ERROR 7344: string expressions may not refer to other columns with string expressions
ERROR 7346: string queries may not refer to a temporary table
ERROR 7347: string queries must refer to tables
ERROR 7368: ALTER NOTIFIER: Unsupported parameter 'string'
ERROR 7371: Cannot alter a column's string when a node is down
ERROR 7372: Cannot assign value to "string" column
ERROR 7394: Columns in COPY may not contain virtual columns
ERROR 7403: describeProjection is not supported in fenced UDx
ERROR 7404: describeTable is not supported in fenced UDx
ERROR 7410: Epsilon must be a non-negative float number and smaller than one-million
ERROR 7421: Expressions on SELECT statements cannot be used in string query definitions
ERROR 7422: External tables cannot have string expressions
ERROR 7430: Initial centers in table 'string' are not distinct
ERROR 7439: Invalid type 'string' for column string
ERROR 7440: Invalid type 'string' for column string of initial_centers_table
ERROR 7442: listProjections is not supported in fenced UDx
ERROR 7444: listTableProjections is not supported in fenced UDx
ERROR 7446: listTables is not supported in fenced UDx
ERROR 7450: max_iterations must be a positive integer and less than one-million
ERROR 7455: MD5 can’t be used in FIPS mode
ERROR 7466: Must specify either a valid initialization method or initial centers table
ERROR 7477: Notifier action is immutable. To set the new action URL you need to drop the notifier, and
           create a new one
ERROR 7486: num_clusters must be a positive integer
ERROR 7487: Number of clusters must not be bigger than the number of rows without null or infinity values
ERROR 7492: Only 'euclidean' is supported for distance_method
ERROR 7496: Prejoin projection unsupported for table string that has default queries or set-using expressions/queries
ERROR 7539: Sequences are not allowed in string expressions of local temp tables
ERROR 7548: Tables with prejoin projections cannot have string expressions
ERROR 7549: Tables with prejoin projections cannot have string queries
ERROR 7550: TableSample does not work with update statements
ERROR 7551: TableSample only works on user defined tables
ERROR 7553: Temporary tables cannot have subqueries as string expressions
ERROR 7558: The response_column is included in the predictor_columns. You should exclude it to have a meaningful model
ERROR 7559: The selected normalization method is not supported. Only support MinMax, zscore, robust_zscore at present
ERROR 7560: The selected outlier detection method is not supported. Only support robust_zscore based outlier detection at present
ERROR 7606: Number of clusters must not be bigger than the number of distinct rows without null or infinity values
ERROR 7634: TableSample is not supported with Aggregate projections
ERROR 7646: Cursor [string] does not support multiple shards
ERROR 7647: Cursor [string] does not support multiple storage containers
ERROR 7648: Cursor [string] does not support this operation
ERROR 7649: Cursor [string] does not support writes
ERROR 7650: describeBlob is not supported in fenced UDx
ERROR 7652: describeFunction does not support function lookup by name / arguments yet
ERROR 7653: describeFunction is not supported in fenced UDx
ERROR 7654: describeType is not supported in fenced UDx
ERROR 7655: describeType is not supported yet
ERROR 7658: listBlobs is not supported in fenced UDx
ERROR 7659: listBlobs is not supported yet
ERROR 7660: listDerivedTables is not supported in fenced UDx
ERROR 7670: UDx cursors do not currently support deleted data
ERROR 7677: Looking up cursor by table is not yet supported
ERROR 7686: ALTER TABLE does not support ADD CONSTRAINT on text index
ERROR 7687: ALTER TABLE does not support ALTER PARTITION on text index
ERROR 7691: DROP PARTITION is not supported on text index
ERROR 7783: ALTER NOTIFIER: MAXMEMORYSIZE cannot be empty
ERROR 7739: Slices are not supported for VMaps
ERROR 7790: Order BY is not allowed within the OVER() clause of User Defined Transforms in projection definitions
ERROR 7792: Resource pool string cannot be modified
ERROR 7818: Unsupported type 'string' for column 'string'
ERROR 7840: Column [string] is duplicated in exclude_columns
ERROR 7841: Column [string] is duplicated in predictor_columns
ERROR 7846: Relation string is empty
ERROR 7848: Table 'string' contains value rows, should be value
ERROR 7867: Live-aggregate projection is not supported for ADD COLUMN ... PROJECTIONS (...)
ERROR 7870: Prejoin projection is not supported for ADD COLUMN ... PROJECTIONS (...)
ERROR 7878: Too many columns in MLA grouping sets: value, while only up to value MLA columns are allowed
ERROR 7889: ALTER MODEL does not support multiple clauses
ERROR 7898: Model string cannot be moved under hcatalog schema string
ERROR 7900: Model cannot be moved to system schema
ERROR 7926: Catalog object string is not either a table or a schema
ERROR 7927: Column "string" is referenced in a column set using expression of projection "string" but is not stored in the projection
ERROR 7930: COMPLEX JOIN without a boolean marker column not supported
ERROR 7931: COMPLEX JOIN without a complex_join_marker() column at the end of the select list not supported
ERROR 7932: COMPLEX JOIN without a subquery not supported
ERROR 7934: Refresh_columns on multiple tables is only supported for "rebuild" mode
ERROR 7935: Refreshed column cannot be any projection's sort key
ERROR 7936: Refreshed column cannot be referenced in any live-aggregate/expression projection
ERROR 7937: Refreshed column cannot be referenced in projection's segmentation expression
ERROR 7938: Refreshed column cannot be referenced in table's partition expression
ERROR 7964: The selected missing value imputation method is not supported. Only support mean and mode based missing value imputation at present
ERROR 7974: string is not supported in fenced U Dx
ERROR 7976: Add column with default subquery is not supported for unsegmented projections
ERROR 7980: Cannot alter model in an incomplete state
ERROR 8008: Refresh_columns with rebuild mode is not supported for unsegmented projections
ERROR 8013: The selected normalization method is not supported. Valid options are 'minmax', 'zscore', and 'robust_zscore'
ERROR 8023: Unsupported column type [string] for column [string]
ERROR 8034: Refresh_columns on enforced PK/Unique constraint columns not supported
ERROR 8038: Cannot alter type of column "string" since it is referenced in the SET USING expression of column "string"
ERROR 8039: Cannot alter type of column with a SET USING expression
ERROR 8040: Cannot refresh column containing grouped containers
ERROR 8043: Refreshed column cannot be any projection's grouped column
ERROR 8060: Cannot COPY "string" from the client using the keyword LOCAL
ERROR 8123: Not enough number of appropriate samples to run the algorithm (excluding samples with NULL, NaN, or Infinity values)
ERROR 8140: Projections on shared storage cannot use explicit node lists
ERROR 8141: Projections on shared storage cannot use explicit range segmentation
ERROR 8155: SET SCHEMA over flex keys tables is not supported
ERROR 8156: SET SCHEMA over text index tables is not supported
ERROR 8180: UDx concurrency request is not supported for empty OVER clause
ERROR 8202: Cannot run refresh_columns or add column with default subquery when any node is recovering
ERROR 8212: Block Dictionary Compression is not supported for data types longer than value bytes
ERROR 8213: Block Dictionary Compression is not supported for data types longer than 65000 bytes
ERROR 8357: Cannot use meta function in partition GROUP BY expression
ERROR 8358: Cannot use non-deterministic function in partition GROUP BY expression
ERROR 8362: PARTITION BY expression must appear at least once in partition GROUP BY expression
ERROR 8363: Partition GROUP BY expression can only reference columns by way of inclusion of the PARTITION BY expression
ERROR 8379: num_clusters must be less than or equal to 10000
ERROR 8386: Cannot use subquery in partition GROUP BY expression
ERROR 8388: PARTITION BY expression may not contain window functions
ERROR 8410: HAVING clauses are not allowed in projections
ERROR 8474: Model string cannot be created under hcatalog schema string
ERROR 8475: Model cannot be created under system schemas
ERROR 8490: Temporary relation is not supported for this function
ERROR 8496: Historical queries are not supported on temporary tables or projections whose underlying tables are temporary
ERROR 8500: Add Column is not supported for unsegmented projections when the node used as initiator is not subscribed to the replica shard
ERROR 8525: Regularization type [string] does not use provided parameter [string], it only affects [string]
ERROR 8534: Cannot grant privileges on system table to PSEUDOSUPERUSER role
ERROR 8535: Cannot grant privileges on system table to SYSMONITOR role
ERROR 8546: Invalid privilege on system table. Only SELECT is supported
ERROR 8558: Feature string only supported in Eon mode
ERROR 8559: Feature string unsupported in Eon mode

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 0A005

This topic lists the errors associated with the SQLSTATE 0A005.
SQLSTATE 0A005 Description
ERRCODE_PLAN_TO_SQL_NOT_SUPPORTED

Error messages associated with this SQLState

ERROR 6679: Cannot generate annotated query when query contains table that does not have a projection
ERROR 6700: Constant NULLs in NOT-IN clause / Nullaware Anti Join not supported for optimizer-generated annotated queries
ERROR 6739: DML not supported for optimizer-generated annotated queries
ERROR 6770: EXPORT not supported for optimizer-generated annotated queries
ERROR 6866: Meta-function not supported for optimizer-generated annotated queries
ERROR 6889: Optimizer can only generate annotated query for SELECT queries
ERROR 6973: System tables not supported for optimizer-generated annotated queries
ERROR 7114: Query with syntactic hints not supported for optimizer-generated annotated queries
ERROR 7132: Under non-default locale, Multi-level aggregates in set operator sub-queries (except UNION ALL) not supported for optimizer-generated annotated queries
ERROR 7187: Multi-level aggregate queries with more than value grouping sets not supported for optimizer-generated annotated queries
ERROR 7449: Match clause not supported for optimizer-generated annotated queries
ERROR 7666: Query with geometry type not supported for optimizer-generated annotated queries
ERROR 7696: Queries with UDX functions with user defined parameter type of string is not supported

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 0B000

This topic lists the errors associated with the SQLSTATE 0B000.

SQLSTATE 0B000 Description
ERRCODE_INVALID_TRANSACTION_INITIATION
Error messages associated with this SQL State

ERROR 2321: Can’t start a Transaction in this context

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQL STATE 0LV01

This topic lists the errors associated with the SQLSTATE 0LV01.

SQLSTATE 0LV01 Description

ERRCODE_INVALID_GRANT_OPERATION

Error messages associated with this SQL State

ERROR 2120: Admin option for a role cannot be granted to string "public"
ERROR 2601: Circular assignation of roles is not allowed
ERROR 3484: Grant option for a privilege cannot be granted to "public"
ERROR 3485: Grant option for a privilege cannot be granted to (and thus revoked from) "public"
ERROR 3486: Grant options cannot be granted back to your own grantor
ERROR 3616: Invalid string statement
ERROR 3719: Invalid option specified for string statement
ERROR 3723: Invalid privilege type "string"
ERROR 3724: Invalid privilege type string for aggregate function
ERROR 3725: Invalid privilege type string for analytic function
ERROR 3726: Invalid privilege type string for database
ERROR 3727: Invalid privilege type string for function
ERROR 3728: Invalid privilege type string for library
ERROR 3729: Invalid privilege type string for procedure
ERROR 3730: Invalid privilege type string for relation
ERROR 3731: Invalid privilege type string for resource pool
ERROR 3732: Invalid privilege type string for schema
ERROR 3733: Invalid privilege type string for sequence
ERROR 3734: Invalid privilege type string for storage location
ERROR 3735: Invalid privilege type string for transform
ERROR 3745: Invalid role name string
ERROR 4056: New string
ERROR 4613: Role "string" cannot be set as default
ERROR 5601: Invalid privilege type string for filter function
ERROR 5602: Invalid privilege type string for parser function
ERROR 5603: Invalid privilege type string for source function
ERROR 6680: Cannot GRANT string authentication to LDAP role
ERROR 6681: Cannot GRANT string authentication to LDAP user
ERROR 6682: Cannot GRANT/REVOKE LDAP role to/from LDAP user or role
ERROR 7163: Cannot materialize schema privileges. Table string is not set to include schema privileges
ERROR 7227: Cannot materialize schema privileges. View string is not set to include schema privileges
ERROR 7894: Invalid privilege type string for model

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22000

This topic lists the errors associated with the SQLSTATE 22000.

SQLSTATE 22000 Description

ERRCODE_DATA_EXCEPTION

Error messages associated with this SQLState

ERROR 3646: Invalid Datum pointer
ERROR 4163: Non-positive value supplied to randomint: value
ERROR 4921: Test Error @string
ERROR 4922: Test Error from @string
ERROR 7474: Non-positive value supplied to randomint_crypto: value
ERROR 8489: Query failed because of an empty result
Error Messages Associated with SQLSTATE 22001

This topic lists the errors associated with the SQLSTATE 22001.

SQLSTATE 22001 Description

ERRCODE_STRING_DATA_RIGHT_TRUNCATION

Error messages associated with this SQLState

ERROR 2991: Date 'string' too long for type string(value)
ERROR 3426: Float 'string' too long for type string
ERROR 3589: Integer 'string' is too long for type string(value)
ERROR 3605: Interval 'string' too long for type string(value)
ERROR 3605: Numeric 'string' is too long for type string
ERROR 3605: Padded octet length (value) exceeds the value octet limit
ERROR 4208: Result (value characters) exceeds the field width (value)
ERROR 4800: String of value octets is too long for type string
ERROR 4800: Time 'string' too long for type string
ERROR 5024: Timestamp 'string' too long for type string
ERROR 5032: TimestampTz 'string' too long for type string
ERROR 5035: Timetz 'string' too long for type string
ERROR 5417: Value too long for type character varying
ERROR 5418: Value too long for type character
ERROR 7401: Date 'string' too long for buffer of length
ERROR 7431: Interval 'string' too long for buffer of length
ERROR 7568: Time 'string' too long for buffer of length
ERROR 7569: Timestamp 'string' too long for buffer of length
ERROR 7570: TimestampTz 'string' too long for buffer of length
ERROR 7571: TimeTz 'string' too long for buffer of length
ERROR 8276: Type string(value) is too short to hold UUID values, need at least value
Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22003

This topic lists the errors associated with the SQLSTATE 22003.

SQLSTATE 22003 Description

ERRCODE_NUMERIC_VALUE_OUT_OF_RANGE

Error messages associated with this SQLState

ERROR 2429: Cannot find matching query in the system
ERROR 2828: Could not convert 'string' to an int8
ERROR 3425: Float "value" is out of range for type string
ERROR 3675: Invalid input for string, exceeds 32 bits: "string"
ERROR 3676: Invalid input for string, exceeds 64 bits: "string"
ERROR 3786: Invalid value for float: "string"
ERROR 4200: Number of buckets must be a positive integer
ERROR 4361: Percentile value must be a number between 0 and 1
ERROR 4704: Sequence exceeded max value
ERROR 4705: Sequence exceeded min value
ERROR 4756: Smoothing factor must be between 0 and 1
ERROR 4795: String "string" is out of range as a float8
ERROR 4796: String "string" is out of range as an int8
ERROR 4845: Sum() overflowed
ERROR 5408: Value "value" is out of range for type string
ERROR 5409: Value "string" is out of range for type int8
ERROR 5411: Value exceeds range of type string
ERROR 5412: Value is too long for type string: "value"
ERROR 6063: Total number of significant digits for value string is more than what is defined. Buffer size is value while actual length of word is value instead
ERROR 7623: Value "string" is out of range for type int64
ERROR 7697: Arithmetic overflow accumulating numeric, operand %Lg
ERROR 7698: Arithmetic overflow adding numerics, operands %Lg, %Lg
ERROR 7699: Arithmetic overflow subtracting numerics, operands %Lg, %Lg
ERROR 7986: Evaluation of expression to be inserted exceeded range of type numeric(value,value)

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22004

This topic lists the errors associated with the SQLSTATE 22004.

SQLSTATE 22004 Description

ERRCODE_NULL_VALUE_NOT_ALLOWED

Error messages associated with this SQLState

ERROR 2110: ACL arrays must not contain null values
ERROR 2501: Cannot set a NOT NULL column (value) to a NULL value in value statement
ERROR 2502: Cannot set a NOT NULL column (string) to a NULL value in INSERT/UPDATE statement
ERROR 2514: Cannot set NOT NULL columns (string) to a NULL value in INSERT/UPDATE statement
ERROR 4195: NULL value detected in data partitioning expression
ERROR 8361: NULL value detected in partition group by expression

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22007

This topic lists the errors associated with the SQLSTATE 22007.
SQLSTATE 22007 Description

ERRCODE_INVALID_DATETIME_FORMAT

Error messages associated with this SQLState

ERROR 2171: AM/PM hour (value) must be between 1 and 12
ERROR 2364: Cannot calculate day of year without year information
ERROR 3439: Format string is invalid for an Interval value
ERROR 3535: Inconsistent use of year value and "BC"
ERROR 3647: Invalid day-of-week 'string'
ERROR 3679: Invalid input syntax for string: "string"
ERROR 3721: Invalid partition key
ERROR 3785: Invalid value for string: "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22008

This topic lists the errors associated with the SQLSTATE 22008.

SQLSTATE 22008 Description

ERRCODE_DATETIME_FIELD_OVERFLOW

Error messages associated with this SQLState

ERROR 2992: Date/time field value out of range: "string"
ERROR 4065: next_day(infinity, DOW) is not defined

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 22009

This topic lists the errors associated with the SQLSTATE 22009.

SQLSTATE 22009 Description

ERRCODE_INVALID_TIME_ZONE_DISPLACEMENT_VALUE

Error messages associated with this SQLState

ERROR 3768: Invalid timezone interval displacement
ERROR 5044: Timezone displacement out of range: "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 2200B

This topic lists the errors associated with the SQLSTATE 2200B.

SQLSTATE 2200B Description

ERRCODE_ESCAPE_CHARACTER_CONFLICT

Error messages associated with this SQLState

ERROR 2699: Conflicting or redundant options
Error Messages Associated with SQLSTATE 2200D

This topic lists the errors associated with the SQLSTATE 2200D.

SQLSTATE 2200D Description

ERRCODE_INVALID_ESCAPE_OCTET

Error messages associated with this SQLState

ERROR 3285: ESCAPE strings must be a single octet, not "value"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22011

This topic lists the errors associated with the SQLSTATE 22011.

SQLSTATE 22011 Description

ERRCODE_SUBSTRING_ERROR

Error messages associated with this SQLState

ERROR 4034: Negative count not allowed
ERROR 4035: Negative length not allowed

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
ERROR 4036: Negative or zero substring start position not allowed
ERROR 4039: Negative substring length not allowed
ERROR 4784: Start position cannot be 0

**Note:** The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

## Error Messages Associated with SQLSTATE 22012

This topic lists the errors associated with the SQLSTATE 22012.

### SQLSTATE 22012 Description

**ERRCODE_DIVISION_BY_ZERO**

### Error messages associated with this SQLState

ERROR 3117: Division by zero

**Note:** The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

## Error Messages Associated with SQLSTATE 22015

This topic lists the errors associated with the SQLSTATE 22015.

### SQLSTATE 22015 Description

**ERRCODE_INTERVAL_FIELD_OVERFLOW**
Error messages associated with this SQLState

ERROR 3606: Interval field value out of range: "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22019

This topic lists the errors associated with the SQLSTATE 22019.

SQLSTATE 22019 Description

ERRCODE_INVALID_ESCAPE_CHARACTER

Error messages associated with this SQLState

ERROR 2729: COPY DELIMITER for column string must be a single character
ERROR 2730: COPY delimiter must be a single character
ERROR 2731: COPY ENCLOSED BY cannot be a whitespace character
ERROR 2732: COPY ENCLOSED BY for column string cannot be a whitespace character
ERROR 2733: COPY ENCLOSED BY for column string must be a single character
ERROR 2734: COPY ENCLOSED BY must be a single character
ERROR 2736: COPY ESCAPE AS for column string must be a single character
ERROR 2737: COPY ESCAPE must be a single character
ERROR 2758: COPY TRIM for column string must be an empty string or a single character
ERROR 2759: COPY trim must be an empty string or a single character
ERROR 3284: ESCAPE strings must be a single character, not "value"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 2201B

This topic lists the errors associated with the SQLSTATE 2201B.

SQLSTATE 2201B Description

ERRCODE_INVALID_REGULAR_EXPRESSION

Error messages associated with this SQLState

ERROR 3742: Invalid regexp match_param: 'character'
ERROR 4552: Regexp match or recursion limit exceeded (rc value)
ERROR 4553: Regexp pattern error at offset value: string
ERROR 4554: Regexp pattern study error: string
ERROR 5064: Too many regular expression subexpressions

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 2201G

This topic lists the errors associated with the SQLSTATE 2201G.

SQLSTATE 2201G Description

ERRCODE_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION

Error messages associated with this SQLState

ERROR 2939: Count must be greater than zero
ERROR 3888: Lower and upper bounds must be finite  
ERROR 3889: Lower bound cannot equal upper bound  
ERROR 4277: Operand, lower bound and upper bound cannot be NaN

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22021

This topic lists the errors associated with the SQLSTATE 22021.

SQLSTATE 22021 Description

ERRCODE_CHARACTER_NOT_IN_REPERTOIRE

Error messages associated with this SQLState

ERROR 4551: Regexp encountered an invalid UTF-8 character

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22023

This topic lists the errors associated with the SQLSTATE 22023.

SQLSTATE 22023 Description

ERRCODE_INVALID_PARAMETER_VALUE
Error messages associated with this SQLState

ERROR 2007: string can not be greater than PASSWORD_MAX_LENGTH value
ERROR 2008: string can not be set to a negative number
ERROR 2028: string exceptions and rejected_data can not be the same filename
ERROR 2042: string must be a positive integer
ERROR 2048: string Path [string] is a directory
ERROR 2049: string Path [string] is a socket
ERROR 2056: string Unrecognized format 'string' for column value
ERROR 2071: 'string' is not a valid size description
ERROR 2075: @INCLUDE without filename in timezone file "string", line value
ERROR 2078: [string] cannot be dropped. There will be no storage locations for data files
ERROR 2079: [string] cannot be retired. There will be no storage locations for temporary files
ERROR 2080: [string] cannot be retired. There will be no storage locations for temporary files
ERROR 2081: [string] is not a valid storage location on node string
ERROR 2108: ACL array contains wrong data type
ERROR 2109: ACL arrays must be one-dimensional
ERROR 2158: All columns of soft unique key statistics must be from the same table
ERROR 2194: analyze_statistics: Can not analyze statistics of a non-local temporary table/projection 'string'
ERROR 2195: analyze_statistics: Can not analyze statistics of a virtual table/projection string
ERROR 2196: analyze_statistics: Cannot analyze statistics of a virtual table string
ERROR 2197: analyze_statistics: invalid accuracy value A number between 0 and 100 is required
ERROR 2254: Bad snapshot name 'string' (cannot contain / or start with a .)
ERROR 2298: Can not lock/unlock super user account
ERROR 2300: Can not reuse any recent passwords
ERROR 2301: Can not reuse current password
ERROR 2302: Can not reuse the previous value passwords
ERROR 2317: Can't purge projection(s); AHI is at epoch 0
ERROR 2319: Can't set a REJECTED file on node 'string', which the current query is not executing on
ERROR 2320: Can't set an EXCEPTIONS file on node 'string', which the current query is not executing on
ERROR 2365: Cannot calculate week number without year information
ERROR 2370: Cannot close a protected session
ERROR 2414: Cannot drop extended statistics on a projection (string)
ERROR 2415: Cannot drop extended statistics on projection string. Dropping base statistics only
ERROR 2452: Cannot load data from node string as it is down
ERROR 2457: Cannot merge partitions in multiple tables at the same time
ERROR 2468: Cannot partition by value multiple tables at the same time
ERROR 2478: Cannot release savepoint; no transaction in progress
ERROR 2500: Cannot set string maxConcurrency to unlimited
ERROR 2508: Cannot set maxConcurrency of string pool to 0
ERROR 2509: Cannot set maxMemorySize of string pool to string [value KB], as it is above 75% [75% = value KB]
ERROR 2510: Cannot set maxMemorySize of string pool to none, as this could prevent moveout from running
ERROR 2511: Cannot set maxMemorySize of recovery pool to string [value KB], as it is below 25% [value KB]
ERROR 2513: Cannot set memorySize of general pool
ERROR 2523: Cannot specify exceptions or rejected-data files ON ANY NODE
ERROR 2540: Cannot use 0 for a key, used internally
ERROR 2548: Cannot use both COPY LOCAL and ON ANY NODE: LOCAL files are stored on the client, not on any Vertica node
ERROR 2621: Collection type must be specified
ERROR 2624: Column "string" does not exist
ERROR 2653: Column string of projection string has ACCESSRANK < 0
ERROR 2695: Conflicting "datestyle" keywords
ERROR 2720: Conversion to timezone "string" failed
ERROR 2722: COPY .. LOCAL cannot store string on a Vertica node
ERROR 2723: COPY ... LOCAL can read files from the client only
ERROR 2724: COPY ... LOCAL can read files with same compression only
ERROR 2727: COPY column option string not supported with format string
ERROR 2728: COPY delimiter string must not appear in the NULL specification
ERROR 2735: COPY ENCLOSEING CHARACTER string must not appear in the NULL specification
ERROR 2748: COPY NULL must be an empty string or a single character for FIXED WIDTH data
ERROR 2749: COPY option string not supported
ERROR 2750: COPY option string not supported with format string
ERROR 2752: COPY RECORD TERMINATOR must be at least ONE character long
ERROR 2753: COPY REJECTMAX should be >= 0
ERROR 2756: COPY skip characters should be >= 0
ERROR 2757: COPY skip should be >= 0
ERROR 2760: COPY WITH PARSER Error (column value): Parser specified a column of type [string]; table needs [string]
ERROR 2761: COPY WITH PARSER Error: Parser specified value column(s); table needs value column(s)
ERROR 2765: COPY: width and length of null string does not match for column string
ERROR 2766: COPY: width for column string has to be greater than 0
ERROR 2830: Could not convert to timezone "string"
ERROR 2932: Couldn’t find the specified task
ERROR 2950: Current design does not meet the requirements for K = value
Current design is valid for K string value
ERROR 2963: CURRENT_TIME(value) precision must not be negative
ERROR 2964: CURRENT_TIME(value) precision reduced to maximum allowed, value
ERROR 2965: CURRENT_TIMESTAMP(value) precision must not be negative
ERROR 2966: CURRENT_TIMESTAMP(value) precision reduced to maximum allowed, value
ERROR 2993: Datepart "string" not recognized
ERROR 2994: Datepart is invalid
ERROR 3006: DDL statement interfered with snapshot; an object no longer exists
ERROR 3007: DDL statement interfered with this statement
ERROR 3012: DECIMAL precision value must be between 1 and value
ERROR 3013: DECIMAL scale value must be between 0 and precision value
ERROR 3032: Delimiter and record terminator cannot be the same value
ERROR 3033: Delimiter and record terminator for string cannot be the same value
ERROR 3137: drop_statistics: Can not drop base or histogram statistics of a non-local temporary table/projection string
ERROR 3138: drop_statistics: Can not drop statistics for a virtual table/projection string
ERROR 3139: drop_statistics: Invalid stats type 'string'. Valid values are 'base', 'histograms' and 'extended'
ERROR 3168: ENCLOSED BY and delimiter string can not be the same value
ERROR 3169: ENCLOSED BY and ESCAPE AS string can not be the same value
ERROR 3170: ENCLOSED BY and record terminator string can not be the same value
ERROR 3178: ENFORCELENGTH cannot be specified for string
ERROR 3280: ESCAPE AS and delimiter string can not be the same value
ERROR 3281: ESCAPE AS and NULL specification string can not be the same value
ERROR 3282: ESCAPE AS and record terminator string can not be the same value
ERROR 3383: Failed to parse object name string
ERROR 3423: Fixed width record size (value) is too large. Record size has to be lesser than value (0xvalue)
ERROR 3424: Fixed width record size is too large. Record size has to be lesser than value (0xvalue)
ERROR 3440: Format cannot be specified for string
ERROR 3503: ICU string error: 'string'
ERROR 3505: ICU does not support locale 'string'
ERROR 3513: Illegal argument to change_runtime_priority: NULL
ERROR 3514: Illegal argument to set_config parameter: NULL
ERROR 3524: In the SAMPLE STORAGE n or SAMPLE STORAGE n,b clause, n must be a constant greater than or equal to 0
ERROR 3525: In the SAMPLE STORAGE n PERCENT or SAMPLE STORAGE n PERCENT,b clause, n must be a constant greater than or equal to 0 and less than or equal to 100
ERROR 3526: In the SAMPLE STORAGE n PERCENT,b clause, n must be a constant greater than or equal to 0 and less than or equal to 100, while b must be a constant greater than or equal to 0
ERROR 3527: In the SAMPLE STORAGE n,b clause, both n and b must be constants greater than or equal to 0
ERROR 3528: In the SAMPLE STORAGE n,b or SAMPLE STORAGE n PERCENT,b clause, b must be a constant greater than or equal to 0
ERROR 3541: Increase in pool size to string [value KB] causes general pool to fall below minimum [25% = value KB]
ERROR 3607: INTERVAL leading field precision increased to value
ERROR 3608: INTERVAL leading field precision reduced to value
ERROR 3610: INTERVAL SECOND precision reduced to value
ERROR 3612: Interval units "value" not recognized
ERROR 3618: Invalid accuracy value for analyze_histogram
ERROR 3632: Invalid collection type string specified
ERROR 3652: Invalid Directives type: string
ERROR 3673: Invalid hint identifier 'string'
ERROR 3686: Invalid interval value for timezone
ERROR 3688: Invalid K value: value K cannot be less than zero
ERROR 3689: Invalid K value: value Maximum K value for value nodes is: value
ERROR 3692: Invalid limit type (string): must be HIGH or LOW
ERROR 3695: Invalid list syntax for "datestyle"
ERROR 3707: Invalid node: [string]
ERROR 3710: Invalid number for timezone offset in timezone file "string", line value
ERROR 3741: Invalid range
ERROR 3743: Invalid resource type (string)
ERROR 3746: Invalid runtime priority string
ERROR 3750: Invalid service name for 'string'
ERROR 3759: Invalid syntax in timezone file "string", line value
ERROR 3767: Invalid timezone file name "string"
ERROR 3777: Invalid Usage type: string
ERROR 3780: Invalid user/role name "string"
ERROR 3818: Invalid value string=string
ERROR 3878: Invalid value for parameter
ERROR 3878: Invalid value for parameter string: string
ERROR 3879: Invalid value for search path: "string"
ERROR 3840: Keyword 'string' (string=string) is not supported
ERROR 3845: Latency should be > 0
ERROR 3852: Length for type string cannot exceed value
ERROR 3853: Length for type string must be at least 1
ERROR 3877: LOCALTIME(value) precision must not be negative
ERROR 3878: LOCALTIME(value) precision reduced to maximum allowed, value
ERROR 3879: LOCALTIMESTAMP(value) precision must not be negative
ERROR 3880: LOCALTIMESTAMP(value) precision reduced to maximum allowed, value
ERROR 3912: maxMemorySize of string [value KB] is not in bounds [max is value KB]
ERROR 3920: memoryCap of string (value KB) would exceed [value KB]
ERROR 3922: memorySize string [value KB] would exceed maxMemorySize string [value KB]
ERROR 3923: memorySize of string [value KB] would exceed [value KB]
ERROR 3960: Missing timezone abbreviation in timezone file "string", line value
ERROR 3961: Missing timezone offset in timezone file "string", line value
ERROR 3967: More than one string specified for a node
ERROR 4027: Must supply a CATALOGPATH
ERROR 4028: Must supply a HOSTNAME
ERROR 4037: Negative run time cap is not allowed
ERROR 4038: Negative runTimeCap is not allowed
ERROR 4089: No objects specified
ERROR 4175: Not allowed to close session
ERROR 4186: NULL is an invalid K value
ERROR 4187: NULL is invalid object name for analyze_extended_statistics
ERROR 4188: NULL is invalid object name for analyze_histogram
ERROR 4189: NULL is invalid object name for drop_statistics
ERROR 4190: NULL is invalid scope type for analyze_extended_statistics
ERROR 4191: NULL is invalid statistics type for analyze_extended_statistics
ERROR 4192: NULL is invalid statistics type for drop_statistics
ERROR 4194: NULL string and record terminator string can not be the same value
ERROR 4211: NUMERIC precision value must be between 1 and value
ERROR 4212: NUMERIC scale value must be between 0 and precision value
ERROR 4222: Occurrence number must be > 0
ERROR 4250: Only ONE exception file should be specified for a LOCAL copy
ERROR 4252: Only ONE rejected data file should be specified for a LOCAL copy
ERROR 4318: Parameter string in default profile can not be set to DEFAULT
ERROR 4319: Parameter string may not exceed 9999
ERROR 4330: PARTITION BY clause must contain table columns in a valid expression
ERROR 4334: Partition key too long
ERROR 4344: PASSWORD_MAX_LENGTH must be within the range from value to value
ERROR 4345: PASSWORD_MIN_DIGITS + PASSWORD_MIN_SYMBOLS + PASSWORD_MIN_LETTERS value can not be greater than PASSWORD_MAX_LENGTH value
ERROR 4346: PASSWORD_MIN_DIGITS + PASSWORD_MIN_SYMBOLS + PASSWORD_MIN_LOWERCASE_LETTERS + PASSWORD_MIN_UPPERCASE_LETTERS value can not be greater than PASSWORD_MAX_LENGTH value
ERROR 4347: Path cannot be an empty string
ERROR 4406: Precision for type float must be at least 1 bit
ERROR 4407: Precision for type float must be less than 54 bits
ERROR 4408: Precision must be less than value; result would be numeric(value,value)
ERROR 4454: Projection string cannot be analyzed, because it is not up to date
ERROR 4456: Projection string cannot drop statistics, because it is not up to date
ERROR 4529: Rebalance skew percent must be in the range [0,100]
ERROR 4556: Regexp starting position must be greater than zero
ERROR 4595: Resource pool "string" is an internal pool and cannot be dropped
ERROR 4606: Retention settings must be less than 2TB
ERROR 4639: Run time cap cannot exceed 1 year
ERROR 4642: runTimeCap cannot exceed 1 year
ERROR 4647: Scaling factor must be greater than zero
ERROR 4648: Scaling factor must be less than 33
ERROR 4653: Schema string is virtual
ERROR 4701: Sequence string is already owned by string
ERROR 4702: SEQUENCE CACHE should be greater than 0
ERROR 4708: SEQUENCE MAXVALUE is too large and will overflow
ERROR 4709: SEQUENCE MINVALUE is too small and will underflow
ERROR 4710: SEQUENCE MINVALUE should be lesser than MAXVALUE
ERROR 4712: SEQUENCE START WITH should be between MINVALUE and MAXVALUE
ERROR 4723: SET string takes only one argument
ERROR 4745: Setting sysdata maxMemorySize below 4 MB to string [value KB] will prevent system table queries from running
ERROR 4766: Specified too few widths for the given number of columns
ERROR 4770: Specify at least one table-column for soft unique key statistics
ERROR 4802: STROKE collations are not supported
ERROR 4807: Subnet mask is empty
ERROR 4862: System pool priority must be between -110 and 110 inclusive
ERROR 4889: Table string is already owned by string
ERROR 4892: Table string is not partitioned
ERROR 4893: Table string is session scoped
ERROR 4894: Table string is virtual
ERROR 4923: That password is not acceptable
ERROR 4937: The confidence level must be between 0 and 100 inclusive. string
ERROR 4961: The permissible error must between 0 and 100 inclusive. string
ERROR 4985: There is no reason to set string.string. Consult documentation
ERROR 5002: Throughput should be > 0
ERROR 5014: Time units "value" not recognized
ERROR 5015: Time units "string" not recognized
ERROR 5019: TIME(value)string precision must not be negative
ERROR 5020: TIME(value)string precision reduced to maximum allowed, value
ERROR 5026: Timestamp units "value" not recognized
ERROR 5027: Timestamp units "string" not recognized
ERROR 5029: TIMESTAMP(value) precision reduced to maximum allowed, value
ERROR 5030: TIMESTAMP(value)string precision must not be negative
ERROR 5031: TIMESTAMP(value)string precision reduced to maximum allowed, value
ERROR 5034: TIMESTAMPTZ(value) precision must not be negative
ERROR 5036: TIMETZ(value) precision must not be negative
ERROR 5037: TIMETZ(value) precision reduced to maximum allowed, value
ERROR 5038: Timezone "string" not recognized
ERROR 5039: Timezone "string" uses leap seconds
ERROR 5041: Timezone abbreviation "string" is multiply defined
ERROR 5042: Timezone abbreviation "string" is too long (maximum value characters) in timezone file "string", line value
ERROR 5045: Timezone file recursion limit exceeded in file "string"
ERROR 5046: Timezone offset value is not a multiple of 900 sec (15 min) in timezone file "string", line value
ERROR 5047: Timezone offset value is out of range in timezone file "string", line value
ERROR 5048: Timezone value "string" is more than value hours
ERROR 5067: Total data collector memory retention of valueKB is too large given system memory size
ERROR 5106: TuningRecommendations data collection is disabled
ERROR 5118: UDL specified no execution nodes; at least one execution node must be specified
ERROR 5136: Unable to log this tuning analysis event
ERROR 5198: Unknown authentication method: "string"
ERROR 5209: Unknown node: string
ERROR 5211: Unknown or unsupported object: string
ERROR 5215: Unknown value string=string
ERROR 5220: Unrecognized "datestyle" keyword: "string"
ERROR 5229: Unrecognized format 'string'
ERROR 5248: Unrecognized privilege type: "string"
ERROR 5258: Unrecognized timezone name: "string"
ERROR 5271: Unsupported format code: value
ERROR 5316: Usage cannot be an empty string
ERROR 5317: Usage of [string] cannot be changed from string to string
ERROR 5319: Usage of [string] cannot be changed to string. There will be no storage locations for data files
ERROR 5320: Usage of [string] cannot be changed to string. There will be no storage locations for temporary files
ERROR 5393: User pool priority must be between -100 and 100 inclusive
ERROR 5437: Vertica should not be run with less than 1GB of RAM
ERROR 5520: string compresses network traffic. string does NOT compress network traffic. Please change the configuration to be consistent
ERROR 5521: string does NOT compress network traffic. string compresses network traffic. Please change the configuration to be consistent
ERROR 5538: Cannot COPY user-defined types directly. Please compute them using copy expressions
ERROR 5542: Cannot INSERT or COPY user-defined types directly. Please compute them using appropriate user-defined functions
ERROR 5545: Cluster layout must include all non-ephemeral nodes and should also not include any ephemeral nodes
ERROR 5549: Conversion from string to DataType string failed. Invalid value
ERROR 5571: Empty storage tier label is not allowed
ERROR 5576: Every permanent node should only be listed once
ERROR 5598: Invalid or unavailable type 'LONG VARBINARY'
ERROR 5599: Invalid or unavailable type 'LONG VARCHAR'
ERROR 5605: Invalid projection createtype 'string'
ERROR 5613: Length for type string must be between 1 and value
ERROR 5631: Object string does not exist or is not of supported type
ERROR 5632: Object string is not a table
ERROR 5634: Path [string] is a directory
ERROR 5644: Projection basename "string" is not a prefix of projection name "string"
ERROR 5645: Projection basename cannot be empty
ERROR 5646: Projection createtype cannot be empty
ERROR 5647: Provided Node "string" does not exist
ERROR 5648: Provided Node "string" is not permanent
ERROR 5668: Target table name can not be empty
ERROR 5685: User Defined Filter expected but found string
ERROR 5686: User Defined Parser expected but found string
ERROR 5687: User Defined Source expected but found string
ERROR 5693: Using 1 year for QUEUETIMEOUT
ERROR 5703: Couldn't find the specified task, or the Resource Manager has not recieved the request
ERROR 5728: Specified too many widths (value) for the given number of columns (value)
ERROR 5740: 'string' is not a valid value for database option string
ERROR 5746: analyze_statistics: invalid number of buckets value. A number > 0 is required
ERROR 5750: Attempt to configure CPU affinity mode conflicts with configuration of resource pool 'string'
ERROR 5751: Attempt to configure CPU affinity set to 'string' conflicts with configuration of resource pool 'string'
ERROR 5752: Attempt to configure CPU affinity set to 'string' in exclusive mode would not leave any CPUs available for system queries
ERROR 5753: Attempt to configure CPU affinity to exclusive mode would not leave any CPUs available for system queries
ERROR 5762: Can only specify user defined file system for DATA and/or TEMP storage locations
ERROR 5767: Cannot do LOCAL and REJECTED DATA AS TABLE in the same query; rejected records can only be saved to one location
ERROR 5768: Cannot do RETURNREJECTED and REJECTED DATA AS TABLE in the same query; rejected records can only be saved to one location
ERROR 5778: Cannot specify both a rejected file and a rejected table in the same statement
ERROR 5779: Cannot specify both an exceptions file and a rejected table in the same statement
ERROR 5793: Control set size out of bounds -1 <= value <= 128
ERROR 5804: CPU #value is not available to this server, because of server-level processor pinning
ERROR 5918: Improperly formatted broadcast address [string]
ERROR 5925: Interface IP address (family string) "string" is invalid
ERROR 5933: Invalid state for UDFilter: REJECT
ERROR 5934: Invalid state for UDSource: INPUT_NEEDED
ERROR 5935: Invalid state for UDSource: REJECT
ERROR 5954: memoryCap of string [value KB] would exceed [value KB]
ERROR 5962: Must request a positive key count to materialize: value
ERROR 5976: Object already exists: string. Can't create a rejections table with the same name
ERROR 6006: Request for value percent of value CPUs rounds to zero CPUs
ERROR 6007: Request for reservation of CPU #value conflicts with another pool's reservation
ERROR 6010: Resource pool 'string' not found
ERROR 6044: Table already exists: string. Can't create a rejections table with the same name
ERROR 6045: The CPU affinity mode cannot be SHARED or EXCLUSIVE if the affinity set is empty
ERROR 6073: Unable to allocate value CPUs for resource pool in string affinity mode
ERROR 6088: Unknown control mode string
ERROR 6090: Unknown database option 'string'
ERROR 6097: User-Defined Load function indicated that it consumed value bytes, when only value were available
ERROR 6123: A Resource Pool cannot cascade to itself
ERROR 6126: A Resource Pool cannot be altered on unknown table "string"
ERROR 6127: Access policy cannot be copied from unknown object "string"
ERROR 6128: Access policy cannot be copied to unknown object "string"
ERROR 6129: Access policy cannot be created on temporary table "string"
ERROR 6130: Access policy cannot be created on unknown table "string"
ERROR 6131: Access policy cannot be created with empty expression
ERROR 6132: Access policy cannot be dropped on unknown object "string"
ERROR 6134: Access policy for COLUMN "string" already exists on "string"
ERROR 6135: Access policy for ROWS already exists on "string"
ERROR 6150: Can not alter access policy on table "string" for COLUMN "string": it doesn't exist
ERROR 6151: Can not alter access policy on table "string" for ROWS: it doesn't exist
ERROR 6152: Can not copy access policy from table "string" for COLUMN "string": it doesn't exist
ERROR 6153: Can not copy access policy from table "string" for ROWS: it doesn't exist
ERROR 6154: Can not drop access policy on table "string" for COLUMN "string": it doesn't exist
ERROR 6155: Can not drop access policy on table "string" for ROWS: it doesn't exist
ERROR 6159: Can't drop indexed column
ERROR 6160: Can't string between these two tables: Text indices don't line up
ERROR 6161: Can't specify a schema for LOCAL TEMP objects
ERROR 6167: Cannot directly modify a text-index table
ERROR 6170: Cannot index external table 'stringstringstring'
ERROR 6172: Cannot move data from a non retired storage location on node string
ERROR 6175: Cannot perform the specified administrative operation on a table with a text index
ERROR 6178: Cannot resolve node address [string] using address family string
ERROR 6179: Cannot resolve node address [string] using family string
ERROR 6180: Cannot resolve node control address [string] using address family string
ERROR 6181: Cannot resolve node control address [string] using family string
ERROR 6189: Client Authentication "string" is disabled
ERROR 6194: Column "string" doesn't exists in "string". Cannot create access policy
ERROR 6202: Correlation STRENGTH must be in the range [-1.0,1.0]
ERROR 6203: Could not access string "string": value
ERROR 6275: General pool priority must be between -100 and 100 inclusive
ERROR 6283: Illegal argument to get_config_parameter: NULL
ERROR 6288: Indexed table must have a projection that is segmented by HASH(string), sorted by string, and contains every column listed in the CREATE TEXT INDEX statement
ERROR 6289: Indexed table must have a projection that is segmented by HASH(string), sorted by string, and contains every column listed in the CREATE TEXT INDEX statement (no projections have been created yet)
ERROR 6295: Invalid declaration; 'tls' method requires HOST TLS
ERROR 6296: Invalid ip address and/or network mask - "string"
ERROR 6297: Invalid number of control nodes value
ERROR 6298: Invalid parameter - "string = string"
ERROR 6299: Invalid resource pool specified for cascade to
ERROR 6382: IP address family conflict: node address family is string, control address family is string
ERROR 6314: LDAP URL [string] seems to be malformed. Client Authentication "string" is disabled
ERROR 6329: Malformed message description
ERROR 6368: Only simple "expression" is allowed in access policy
ERROR 6383: Path "string" exists, but it is directory, not a file
ERROR 6384: Path "string" exists, but it is not a directory
ERROR 6387: Projection replace oid is empty or malformed
ERROR 6397: Sequences cannot be used in access policy
ERROR 6414: Subnet IP address (family string) "string" is invalid
ERROR 6421: Table name can not be empty
ERROR 6456: Unsupported authentication method - "string"
ERROR 6473: You cannot use both parameters 'string' and 'string'. Client Authentication 'string' is disabled
ERROR 6475: Can't swap partitions between these two tables: Text indices don't line up
ERROR 6492: No text index column named "string". Indexed text column in text index table must exist with name "string"
ERROR 6504: Bad expression for Access Policy
ERROR 6509: Cascade to cannot be used to make a resource pool loop
ERROR 6519: Bad expression for Access Policy - "string"
ERROR 6520: Can not copy access policy from table "string" for COLUMN "string": "string"
ERROR 6521: Can not copy access policy from table "string" for ROWS: "string"
ERROR 6545: NULL is invalid object name for analyze_external_row_count
ERROR 6546: NULL is invalid object name for drop_external_row_count
ERROR 6566: Invalid constant hint:
ERROR 6569: Invalid table hint identifier 'string'
ERROR 6609: string process has reserved resources. Release it first
ERROR 6619: /*+syntactic_join*/ hint omitted, ignoring join hints
ERROR 6641: Argument of nth_value must be greater than zero
ERROR 6661: Can not expire password on LDAP user account
ERROR 6663: Can not modify password on LDAP user account
ERROR 6664: Can not modify profile on LDAP user account
ERROR 6666: Can not set Security Algorithm on LDAP user account
ERROR 6689: Cannot store TEMP data on HDFS storage locations
ERROR 6802: Hint _oidref is empty or malformed
ERROR 6803: Idemp authentication not allowed for remote connection types
ERROR 6824: Invalid explain hint identifier 'string'
ERROR 6825: Invalid join hint identifier 'string'
ERROR 6835: Invalid with hint identifier 'string'
ERROR 6934: Resource Allocation for this process not found
ERROR 6944: Schema "string" is already set to string privileges
ERROR 6979: Table string does not contain a key constraint 'string'
ERROR 6980: Table string is already set to inherit privileges
ERROR 6981: Table string is already set to not inherit privileges
ERROR 7042: Unknown node
ERROR 7060: User string could not be found in the update list
ERROR 7061: User-Defined Load function indicated that it consumed value bytes from the record lengths buffer, when only value were available
ERROR 7069: View "string" is already set string inherit privileges
ERROR 7083: */+syntactic_join*/ hint omitted, ignoring union hints
ERROR 7102: Invalid union hint identifier 'string'
ERROR 7104: Length of string type function argument cannot exceed value [string]
ERROR 7108: Parameter 'string' cannot be NULL
ERROR 7140: DataBuffer indicated that it read value bytes, while LengthBuffer indicated that value data bytes were read
ERROR 7141: DataBuffer indicated that it wrote value bytes, while LengthBuffer indicated that value data bytes were written
ERROR 7145: Invalid end-of-file state for string: INPUT_NEEDED
ERROR 7146: Invalid state for UDFParser: OUTPUT_NEEDED
ERROR 7154: UDChunker indicated that it consumed bytes, but returned INPUT_NEEDED
ERROR 7156: Access policy cannot be created on system table "string"
ERROR 7159: Backup type is invalid
ERROR 7160: Cannot expand glob pattern due to error: string
ERROR 7162: Cannot load data due to error: string
ERROR 7168: Column Access Policies on flex tables may not be completely secure
ERROR 7174: Cyclic access policy detected for relation: string
ERROR 7179: ERROR TOLERANCE is not supported for string loads
ERROR 7180: ERROR TOLERANCE is not supported for ORC files
ERROR 7184: Illegal argument to create_storage_containers: NULL
ERROR 7185: Illegal argument: Please enter positive number for creating ROS containers
ERROR 7189: Occurrence parameter must be non-negative
ERROR 7194: Restore type is invalid
ERROR 7209: View string is already owned by string
ERROR 7225: Backup epoch [value] must be earlier than the current epoch [value]
ERROR 7240: Constraint 'string' on table string is not a primary or unique key, nor a check constraint
ERROR 7254: ERROR TOLERANCE is not supported for Parquet files
ERROR 7269: Invalid portion: string [value]
ERROR 7271: Negative queueTimeout is not allowed
ERROR 7304: Cannot load data from [string]: all specified nodes are down
ERROR 7306: Cannot specify exceptions or rejected-data files on multiple nodes
ERROR 7312: Column string does not exist in Table string
ERROR 7313: Column string in Table string does not have SET USING expression
ERROR 7321: Invalid action string specified
ERROR 7322: Invalid pool name string specified
ERROR 7334: Size $value$ out of range
ERROR 7335: Table $string$ does not have any column with a SET USING expression
ERROR 7387: Cannot remove session's idle timeout
ERROR 7417: Error in blob creation: nChunks must be greater than 0
ERROR 7432: Invalid action URI '{$string'}': adapter not supported
ERROR 7434: Invalid cursor request source: $value$
ERROR 7436: Invalid number of arguments
ERROR 7451: Maxconnections cannot be greater than total number of allowable connections
ERROR 7452: Maxconnections cannot be set for a superuser
ERROR 7453: Maximum message size for notifier cannot be less than $value$
ERROR 7454: Maximum message size for notifier cannot be more than $value$
ERROR 7456: Memory size cannot be zero
ERROR 7459: Message cannot be empty
ERROR 7465: Must set mode in which connection limit is applicable
ERROR 7467: Negative idlesessiontimeout is not allowed
ERROR 7468: New idlesessiontimeout $string$ would exceed database wide limit of $string$
ERROR 7469: New idlesessiontimeout $string$ would exceed user limit of $string$
ERROR 7472: No channel is set
ERROR 7473: No notifier is set
ERROR 7476: Notifier "$string$" does not exist
ERROR 7481: Notifier memory size ($value$) cannot be less than the maximum message size ($value$)
ERROR 7482: Notifier memory size ($value$) should be less than 2TB
ERROR 7491: Object name cannot be null
ERROR 7536: Sample Count should be between 0 and 100
ERROR 7541: Skipnode hint needs node name
ERROR 7542: Skipnode hint omitted in sub-query
ERROR 7552: Temporary data cursor request requires type information
ERROR 7564: There is no node with such name: [string]
ERROR 7578: UDSource requested 0 threads on node [string] which is targeted for execution
ERROR 7584: Using 1 year as idlesessiontimeout
ERROR 7587: Valid value for outlierThreshold is a positive float number
ERROR 7588: Valid value for sampling_ratio is a positive float number between 0 and 1
ERROR 7615: Requested too many nodes
ERROR 7624: When an empty map of nodes is supplied, expect nNodes > 0
ERROR 7674: Idlesessiontimeout cannot be less than 15 minutes for superuser. Using 15 minutes as idlesessiontimeout
ERROR 7707: Chunk index $value$ out of range 0-$value$
ERROR 7714: Error in blob creation: maxSize cannot be negative
ERROR 7715: Error in blob creation: minSize cannot be negative
ERROR 7716: Error in blob creation: minSize must be less than maxSize
ERROR 7727: Invalid notifier adapter configuration: $string$
ERROR 7741: Tried to read ($value$) past end of chunk ($value$)
ERROR 7742: Tried to read (at value) past end of chunk
ERROR 7749: Unknown notifier adapter
ERROR 7753: NULL is invalid accuracy value for analyze_histogram
ERROR 7754: NULL is invalid column name for analyze_histogram
ERROR 7770: miniBatch must be a positive integer
ERROR 7771: No resource grant exists with this id
ERROR 7799: Connection address family 'string' is invalid; expected one of: 'ipv4','ipv6'
ERROR 7800: Connection address family 'string' is not valid for host IP address 'string'
ERROR 7868: Must specify shared storage for built-in shared file system
ERROR 7871: Projection "string" refers to column "string" referenced in the added column's default expression
ERROR 7877: Target table "string" is not anchor table for Projection "string"
ERROR 7899: Model string is already owned by string
ERROR 7910: Invalid column string definition for column string
ERROR 7911: Snapshot type is invalid
ERROR 7918: Invalid load stack: string
ERROR 7919: Invalid state for UDChunker::string(): string
ERROR 7924: UDChunker aligned a 0-byte chunk; chunks must be non-empty
ERROR 7933: Invalid refresh columns mode "string". Specify either "update" or "rebuild"
ERROR 7993: No table specified
ERROR 8065: Cannot load data from node string as it is not in the elastic throughput group selected for this session
ERROR 8068: Cannot remove session's grace period
ERROR 8073: Column string must be qualified as multiple tables are specified
ERROR 8087: Error in blob creation: name cannot be empty
ERROR 8097: Invalid path string for file system string
ERROR 8098: Invalid projection name 'string' while parsing an optimizer directive
ERROR 8104: LIMIT can't be specified for non depot locations
ERROR 8110: Minimum value for string is string
ERROR 8160: Size cannot be an empty string
ERROR 8167: Table string does not exist for specified column string
ERROR 8168: Table string does not have any column in the specified column list
ERROR 8169: Table string for column string is not specified in the table list
ERROR 8175: The new period string would exceed the string limit of string
ERROR 8187: Using string (maximum) as string
ERROR 8190: 'directory' parameter cannot be empty
ERROR 8191: 'directory' parameter must be specified
ERROR 8192: Compression 'string' is invalid
ERROR 8193: Directory [string] exists. Please remove it or specify another directory
ERROR 8195: RowgroupSizeMB 'value' is invalid. Must be (1 MB - 512 MB)
ERROR 8196: Temporary directory [string] exists. Please retry the query
ERROR 8197: Unable to look up UDFS for File System [string]
ERROR 8198: Unable to verify if directory [string] exists due to 'string'
ERROR 8199: Unable to verify if temporary directory [string] exists due to 'string'
ERROR 8215: 'compression' value must be of string type
ERROR 8216: 'directory' value must be of string type
ERROR 8217: 'rowGroupSizeMB' value must be of Integer type
ERROR 8218: 'tempsuffix' value must be of string type
ERROR 8245: Cannot create subscription as PRIMARY
ERROR 8246: Cannot create subscription in string state
ERROR 8247: Cannot drop a subscription for DOWN node string
ERROR 8249: Cannot make a subscription for DOWN node string as PRIMARY
ERROR 8250: Invalid node or shard name
ERROR 8253: Catalog name must not be specified in object names: string
ERROR 8254: Invalid or empty object name: string
ERROR 8285: Missing parameter: parentName. Please provide parentName parameter
ERROR 8286: Missing parameter: parentType. Please provide parentType parameter
ERROR 8294: Cannot identify the type of [string]
ERROR 8297: Invalid JSON string [string]
ERROR 8302: Parameter check failed: string
ERROR 8303: Parameter check warning: string
ERROR 8402: Cannot create subscription for "string" when node is in EXECUTE state
ERROR 8411: Hyper parameter [string] is empty
ERROR 8412: Hyper parameter [string] is invalid
ERROR 8415: Metric [string] is not supported for algorithm [string]
ERROR 8425: Regularization type [string] is not supported for given optimizer [string]. Use [string]
ERROR 8433: The "count" member of JSON object [string] must be a positive integer
ERROR 8434: The "first" and "step" members of JSON object [string] must be numbers
ERROR 8452: Invalid file path: string
ERROR 8457: Usage type not supported. Only COMMUNAL and USER locations are supported for S3 file system
ERROR 8476: Usage:
   Requires a Service Type followed by the Service Name as found by calling list_services
   Supported Service Types: 'TM'
   Example:
   select disable_service('TM','TM Moveout');
ERROR 8477: Usage:
   Requires a Service Type followed by the Service Name as found by calling list_services
   Supported Service Types: 'TM'
   Example:
   select enable_service('TM','TM Moveout');
ERROR 8478: Usage:
   Requires a Service Type followed by the Service Name as found by calling list_services
   Supported Service Types: 'TM', 'SYSTEM'
   Example:
   select hurry_service('TM','TM Moveout');
ERROR 8479: Usage:
   Requires a Service Type to list services
   Supported Service Types: 'TM', 'SYSTEM'
Example:
   select list_services('TM');
ERROR 8488: Node string is not valid
ERROR 8494: Usage of S3 location [string] cannot be changed
ERROR 8502: Cannot drop subscription for replica shard
ERROR 8505: Catalog name must not be specified in wildcard: string
ERROR 8514: Intercept Mode [string] cannot be used when pre-centering the data. Use [string]
ERROR 8520: string could not create directory for string 'string'
ERROR 8532: string input file and string can not be the same file [string]
ERROR 8533: string path for string ['string'] must be a directory for multiple sources
ERROR 8549: Schema string is session scoped

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22025

This topic lists the errors associated with the SQLSTATE 22025.

SQLSTATE 22025 Description

ERRCODE_INVALID_ESCAPE_SEQUENCE

Error messages associated with this SQLState

ERROR 3656: Invalid escape sequence
ERROR 3657: Invalid escape string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 22906

This topic lists the errors associated with the SQLSTATE 22906.

SQLSTATE 22906 Description

ERRCODE_NONSTANDARD_USE_OF_ESCAPE_CHARACTER

Error messages associated with this SQLState

ERROR 4166: Nonstandard use of \ in a string literal at or near "string"
ERROR 4167: Nonstandard use of \ in a string literal at or near "string"
ERROR 4168: Nonstandard use of escape in a string literal at or near "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22V02

This topic lists the errors associated with the SQLSTATE 22V02.

SQLSTATE 22V02 Description

ERRCODE_INVALID_TEXT_REPRESENTATION

Error messages associated with this SQLState

ERROR 2825: Could not convert "string" to a boolean
ERROR 2826: Could not convert "string" to a float8
ERROR 2827: Could not convert "string" to an int8
ERROR 3677: Invalid input for string: "string"
ERROR 3680: Invalid input syntax for boolean: "string"
ERROR 3681: Invalid input syntax for integer: "string"
ERROR 3682: Invalid input syntax for numeric: "value"
ERROR 3711: Invalid number: "string"
ERROR 3712: Invalid numeric format string
ERROR 3714: Invalid numeric value: "string"
ERROR 3751: Invalid Session ID format
ERROR 3757: Invalid syntax for float: "string"
ERROR 3758: Invalid syntax for numeric: "string"
ERROR 3894: Malformed record literal: "string"
ERROR 4169: Not a number: "string"
ERROR 4198: Number exceeds format: "string"
ERROR 5930: Invalid numeric format string. Expected precision is value and scale is value
ERROR 8264: Invalid input syntax for type string: "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22V03

This topic lists the errors associated with the SQLSTATE 22V03.

SQLSTATE 22V03 Description

ERRCODE_INVALID_BINARY_REPRESENTATION

Error messages associated with this SQLState

ERROR 2829: Could not convert integer valuestring to a boolean
ERROR 3536: Incorrect binary data format in bind parameter value
ERROR 3623: Invalid binary input syntax: 'value'
ERROR 3624: Invalid bitstring "string"
ERROR 3671: Invalid hex string "string"
ERROR 3678: Invalid input syntax for string
ERROR 3716: Invalid octal string format "string"
Error Messages Associated with SQLSTATE 22V04

This topic lists the errors associated with the SQLSTATE 22V04.

SQLSTATE 22V04 Description

ERRCODE_BAD_COPY_FILE_FORMAT

Error messages associated with this SQLState

ERROR 2031: string Header size (value) is corrupted
ERROR 2032: string Header size (value) is too small
ERROR 2035: string Input record value has been rejected (string)
ERROR 2053: string Row size (value) is corrupted
ERROR 2054: string Unexpected EOF while reading header. Expected value but read value
ERROR 2738: COPY file signature not recognized
ERROR 2767: COPY: Wrong Header size value. Expected value
ERROR 3562: Input has extra trailing bytes
ERROR 3640: Invalid COPY file header (unsupported Version Number)
ERROR 4206: Number of fields is value, expected value
ERROR 4627: Row delimiter not found; corrupt file input (read value bytes from input)
ERROR 5495: Wrong size value for bool column value (string)
ERROR 5496: Wrong size value for date column value (string)
ERROR 5497: Wrong size value for float column value (string)
ERROR 5498: Wrong size value for integer column value (string)
ERROR 5499: Wrong size value for Interval column value (string)
ERROR 5500: Wrong size value for Numeric column value (string)
ERROR 5501: Wrong size value for Time column value (string)
ERROR 5502: Wrong size value for Timestamp column value (string)
ERROR 5503: Wrong size value for TimestampTz column value (string)
ERROR 5504: Wrong size value for TimeTz column value (string)
ERROR 6363: Not able to skip value rows for source [string]
ERROR 7293: string [value] records have been rejected
ERROR 8394: Wrong size value for Uuid column value (string)

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22V0B

This topic lists the errors associated with the SQLSTATE 22V0B.

SQLSTATE 22V0B Description

ERRCODE_ESCAPE_CHARACTER_ON_NOESCAPE

Error messages associated with this SQLState

ERROR 2746: COPY NO ESCAPE cannot also contain an ESCAPE clause
ERROR 2747: COPY NO ESCAPE for column string cannot also contain an ESCAPE clause

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22V21

This topic lists the errors associated with the SQLSTATE 22V21.
SQLSTATE 22V21 Description

ERRCODE_INVALID_EPOCH

Error messages associated with this SQLState

ERROR 2144: AHM can't advance past the cluster last full backup epoch. (Last Backup Epoch: value)
ERROR 2145: AHM can't advance past the cluster last backup time. (Last Backup time: string)
ERROR 2146: AHM can't advance past the cluster last good epoch (LGE) time (Cluster LGE time: string)
ERROR 2147: AHM can't advance past the cluster last good epoch (LGE). (Cluster LGE: value)
ERROR 2148: AHM can't advance past the latest epoch time (Latest epoch time: string)
ERROR 2153: AHM must be less than the current epoch (Current Epoch: value)
ERROR 2318: Can't run historical queries at epochs prior to the Ancient History Mark
ERROR 3000: DDL interfered with this statement
ERROR 3184: Epoch specified is not in historical epoch range
ERROR 3559: Input epoch must be greater than or equal to the earliest epoch (earliest epoch: value)
ERROR 3560: Input epoch must be greater than the current AHM (Current AHM: value)
ERROR 3561: Input epoch must be less than or equal to the AHM epoch (AHM epoch: value)
ERROR 3567: Input time can't be rounded down to an epoch higher than the current AHM epoch (Current AHM epoch: value, Current AHM time: string)
ERROR 3568: Input time must be greater than or equal to the earliest epoch time (Earliest epoch time: string)
ERROR 3569: Input time must be greater than the current AHM time (Current AHM time: string)
ERROR 3570: Input time must be less than or equal to the AHM epoch time (AHM epoch time: string)
ERROR 3654: Invalid epoch
ERROR 3844: Last good epoch not set
ERROR 3926: MergeOut start epoch (=value) greater than end epoch (=value)
ERROR 4940: The current AHM is already value
ERROR 5013: Time specified is not in historical epoch range
ERROR 7639: AHM can't advance due to lack of full backup

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 22V23

This topic lists the errors associated with the SQLSTATE 22V23.

SQLSTATE 22V23 Description

ERRCODE_RAISE_EXCEPTION

Error messages associated with this SQLState

ERROR 5783: Client error: string (in function string() at string:value)
ERROR 7084: ABORTRECOVERY is only allowed in UNSAFE mode
ERROR 7136: USER GENERATED ERROR
ERROR 7137: USER GENERATED ERROR: value
ERROR 7804: LGE is lagging behind for value seconds, it exceeds value seconds threshold, LGE: value, current epoch: value
ERROR 7813: Snapshot failed because some projections unable to move out data to ROS. Minimum checkpoint epoch of projections in snapshot: value, current epoch: value

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 22V24

This topic lists the errors associated with the SQLSTATE 22V24.

SQLSTATE 22V24 Description

ERRCODE_COPYPARSE_ERROR
Error messages associated with this SQLState

ERROR 6703: Corrupt orc source!
ERROR 6723: Data type string is not supported for the ORC file format
ERROR 6726: Datatype mismatch: column value in the orc source [string] has type string, expected string
ERROR 6777: Failed to read orc source [string]: string
ERROR 7157: Attempt to load value columns from an orc source [string] that has value columns and value partition columns
ERROR 7223: Attempt to load value columns from a parquet source [string] that has value columns and value partition columns
ERROR 7257: Failed to read parquet source [string]
ERROR 7258: Failed to read parquet source [string]: string
ERROR 7259: Failed to read parquet source [string]: complex types are not supported
ERROR 7286: The table has value columns, but the parquet source [string] has value columns and value partition columns
ERROR 7985: Decimal type with precision value (> 38) is not supported

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 23502

This topic lists the errors associated with the SQLSTATE 23502.
SQLSTATE 23502 Description

ERRCODE_NOT_NULL_VIOLATION

Error messages associated with this SQLState

ERROR 2416: Cannot drop NOT NULL constraint on column "string" when it is referenced in PARTITION BY expression
ERROR 2417: Cannot drop NOT NULL constraint on column "string" when it is referenced in primary key constraint
ERROR 2623: Column "string" definition changed to NOT NULL
ERROR 4182: NOT NULL constraint on column "string" already exists
ERROR 4183: NOT NULL constraint on column "string" does not exist

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 23503

This topic lists the errors associated with the SQLSTATE 23503.

SQLSTATE 23503 Description

ERRCODE_FOREIGN_KEY_VIOLATION

Error messages associated with this SQLState

ERROR 4165: Nonexistent foreign key value detected in FK-PK join [string]; value [string]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 23505

This topic lists the errors associated with the SQLSTATE 23505.

SQLSTATE 23505 Description
ERRCODE_UNIQUE_VIOLATION

Error messages associated with this SQLState

ERROR 3147: Duplicate MERGE key detected in join [string]; value [string]
ERROR 3149: Duplicate primary/unique key detected in join [string]; value [string]
ERROR 4840: Subquery used as an expression returned more than one row
ERROR 6744: Duplicate key values in table 'string': 'string' -- violates constraint 'string'
ERROR 6745: Duplicate key values: 'string' -- violates constraint 'string'
ERROR 7695: Null value in primary key column: 'string' -- violates constraint 'string'

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 23514

This topic lists the errors associated with the SQLSTATE 23514.

SQLSTATE 23514 Description
ERRCODE_CHECK_VIOLATION
Error messages associated with this SQLState

ERROR 7230: Check constraint 'value' value violation: 'value'
ERROR 7231: Check constraint 'string' string violation in table 'string': 'string'
ERROR 7232: Check constraint 'string' string violation: 'string'

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 25000

This topic lists the errors associated with the SQLSTATE 25000.

SQLSTATE 25000 Description

ERRCODE_INVALID_TRANSACTION_STATE

Error messages associated with this SQLState

ERROR 8058: Active subscriptions changed during query planning
ERROR 8231: Invalid number of shards to operate on => value
ERROR 8233: Node is not the primary for shard (value)
ERROR 8269: Node is no longer the primary for shard (value)
ERROR 8298: Multiple shards when partitioning ROSs
ERROR 8383: Node is not the primary for shard (value)
ERROR 8568: The shard (value) is not in the active shard set of this mergeout session

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 25V01

This topic lists the errors associated with the SQLSTATE 25V01.

SQLSTATE 25V01 Description

ERRCODE_NO_ACTIVE_SQL_TRANSACTION

Error messages associated with this SQLState

ERROR 2342: Cannot advance epoch without a transaction

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 28000

This topic lists the errors associated with the SQLSTATE 28000.

SQLSTATE 28000 Description

ERRCODE_INVALID_AUTHORIZATION_SPECIFICATION

Error messages associated with this SQLState

ERROR 2701: Conflicting, redundant or unsupported option: string
ERROR 2702: Conflicting, redundant or unsupported option: groupElts
ERROR 2959: Current user cannot be dropped
ERROR 4293: Option "string" not recognized
ERROR 4722: Session user cannot be dropped
ERROR 4846: Superuser cannot be dropped
ERROR 5387: User does not exist
ERROR 6441: Trying to change password, but user "string" does not exist
ERROR 6442: Trying to find authentication method, but user "string" does not exist
ERROR 6443: Trying to update login history, but user "string" does not exist
ERROR 6494: SecurityAlgorithm valid options are NONE, MD5 and SHA512
ERROR 6671: Cannot change password on LDAP user "string"
ERROR 6841: LDAP role "string" cannot be dropped unless it is orphaned (sourced from a different LDAP than currently configured)
ERROR 6842: LDAP user "string" cannot be renamed
ERROR 6843: LDAP user "string" has not database password
ERROR 6844: LDAP user cannot be dropped unless it is orphaned
ERROR 7667: SecurityAlgorithm valid options are NONE and SHA512

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 2BV01

This topic lists the errors associated with the SQLSTATE 2BV01.

SQLSTATE 2BV01 Description

ERRCODE_DEPENDENT_OBJECTS_STILL_EXIST

Error messages associated with this SQLState

ERROR 3052: Dependent privileges exist
ERROR 3128: DROP failed due to dependencies
ERROR 3130: DROP PROFILE failed due to dependencies
ERROR 3131: DROP ROLE failed due to dependencies
ERROR 6240: DROP AUTHENTICATION failed due to dependencies
ERROR 7303: Cannot drop table "string" because other objects depend on it
ERROR 7345: string failed due to dependencies
ERROR 7945: Column string is dropped which will break some dependent access policies on the table. Certain queries may fail
Error Messages Associated with SQLSTATE 38004

This topic lists the errors associated with the SQLSTATE 38004.

SQLSTATE 38004 Description

ERRCODE_E_R_E_READING_SQL_DATA_NOT_PERMITTED

Error messages associated with this SQLState

ERROR 7339: User may not have read privilege to the external table storage location: [ string ]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 40V01

This topic lists the errors associated with the SQLSTATE 40V01.

SQLSTATE 40V01 Description

ERRCODE_T_R_DEADLOCK_DETECTED

Error messages associated with this SQLState

ERROR 3010: Deadlock: string - string
ERROR 3011: Deadlock: [Txn value] string - string error string
Error Messages Associated with SQLSTATE 42501

This topic lists the errors associated with the SQLSTATE 42501.

SQLSTATE 42501 Description

ERRCODE_INSUFFICIENT_PRIVILEGE

Error messages associated with this SQLState

ERROR 2065: string: Invalid table/projection/column string
ERROR 2198: analyze_statistics: Requires modify permissions for table/projection/column string
ERROR 2347: Cannot alter predefined role "string"
ERROR 2348: Cannot alter superuser string's default roles
ERROR 2349: Cannot alter superuser roles
ERROR 2389: Cannot alter predefined role
ERROR 2481: Cannot remove memoryCap
ERROR 2482: Cannot remove runTimeCap
ERROR 2484: Cannot remove tempSpaceCap
ERROR 2515: Cannot set resource pool: user string lacks privileges on resource pool string
ERROR 2812: Could not add location [string]: Permission denied
ERROR 2935: Couldn't nice(value) thread: value
ERROR 2953: Current password must be supplied to set new password
ERROR 2958: Current user can't change runtime priority of another user's task
ERROR 2960: Current user doesn't have the privilege to change the task runtime priority to be higher than its resource pool
ERROR 3577: Insufficient permissions on projection "string"
ERROR 3578: Insufficient permissions on schema "string"
ERROR 3579: Insufficient permissions on table "string"
ERROR 3580: Insufficient privilege: USAGE on SCHEMA 'string' not granted for current user
ERROR 3581: Insufficient privileges for projection string
ERROR 3582: Insufficient privileges for table string
ERROR 3583: Insufficient privileges on string
ERROR 3584: Insufficient privileges on string, modify privileges (INSERT|UPDATE|DELETE|TRUNCATE) needed
ERROR 3722: Invalid passphrase: string
ERROR 3919: memoryCap of value KB would exceed user limit of value KB
ERROR 3989: Must be owner of string string
ERROR 3990: Must be owner of string [string]
ERROR 3991: Must be superuser to alter database
ERROR 3992: Must be superuser to alter profile
ERROR 3993: Must be superuser to alter tuning rule
ERROR 3994: Must be superuser to alter user default roles
ERROR 3998: Must be superuser to clear Query/EE profiles
ERROR 4000: Must be superuser to create interface
ERROR 4001: Must be superuser to create library
ERROR 4002: Must be superuser to create profile
ERROR 4003: Must be superuser to create subnet
ERROR 4004: Must be superuser to create tuning rule
ERROR 4005: Must be superuser to create users
ERROR 4006: Must be superuser to drop an interface
ERROR 4007: Must be superuser to drop library
ERROR 4008: Must be superuser to drop profile
ERROR 4009: Must be superuser to drop resource pool
ERROR 4010: Must be superuser to drop role
ERROR 4011: Must be superuser to drop subnet
ERROR 4012: Must be superuser to drop tuning rule
ERROR 4013: Must be superuser to drop users
ERROR 4014: Must be superuser to modify resource pools
ERROR 4015: Must be superuser to rename interface
ERROR 4016: Must be superuser to rename profile
ERROR 4017: Must be superuser to rename role
ERROR 4018: Must be superuser to rename subnet
ERROR 4019: Must be superuser to run string
ERROR 4020: Must be superuser to run analyze_workloadstring()
ERROR 4059: New runTimeCap value ms would exceed user limit of value ms
ERROR 4061: New tempSpaceCap value KB would exceed user limit of value KB
ERROR 4178: Not enough privileges for projection string
ERROR 4179: Not enough privileges for table string
ERROR 4244: Only database superuser can drop procedures
ERROR 4260: Only superuser can check privileges on other users
ERROR 4261: Only superuser can create roles
ERROR 4269: Only the database super user can create procedures
ERROR 4366: Permission denied
ERROR 4367: Permission denied for string string
ERROR 4368: Permission denied for string [string]
ERROR 4369: Permission denied to create temporary tables
ERROR 4370: Permission denied: "string" is a system catalog
ERROR 4546: RecvFiles on string: Can't write to file [string]
ERROR 4741: setThreadCPUNiceValue: couldn't nice(value) thread: value
ERROR 4742: setThreadIONiceValue: couldn't ionice(value) thread: value
ERROR 5149: Unable to set role "string"
ERROR 5389: User has insufficient privileges on schema string
ERROR 5488: Workspace schema string does not exist
ERROR 5517: Your Vertica license is invalid or has expired
ERROR 5618: Must be superuser to alter fault group
ERROR 5619: Must be superuser to create fault group
ERROR 5620: Must be superuser to drop fault group
ERROR 5635: Path to file [string] contains a symbolic link
ERROR 5716: Must have create permissions in schema string to drop type
ERROR 5818: Deployment script will not be generated since the user does not have appropriate permissions to write to [string]
ERROR 5820: Design script will not be generated since the user does not have appropriate permissions to write to [string]
ERROR 5956: Must be superuser to ALTER NODE
ERROR 5957: Must be superuser to create filesystem
ERROR 5958: Must be superuser to create location
ERROR 5959: Must be superuser to CREATE NODEs
ERROR 5961: Must be superuser to supply 'user_name' argument to HAS_ROLE() function
   HINT: Non-superusers run HAS_ROLE("role_name")
ERROR 5975: Not enough privileges for string
ERROR 6125: Access Policies are not enabled
ERROR 6343: Must be superuser to string authentication
ERROR 6344: Must be superuser to alter access policy
ERROR 6345: Must be superuser to alter authentication
ERROR 6346: Must be superuser to copy access policy
ERROR 6347: Must be superuser to create access policy
ERROR 6348: Must be superuser to create authentication
ERROR 6349: Must be superuser to drop access policy
ERROR 6350: Must be superuser to drop authentication
ERROR 6351: Must be superuser to rename authentication
ERROR 6514: Must be superuser to run value <name not available>
ERROR 6540: string: Invalid external table string
ERROR 6871: Must be superuser to rename user
ERROR 6872: Must be superuser to run move_statement_to_resource_pool
ERROR 6896: Parameter string can be cleared only via a UDx
ERROR 6897: Parameter string can be set only via a UDx
ERROR 6902: Permission denied on [string.string]
ERROR 6903: Permission denied, must be superuser to create a directed query
ERROR 6904: Permission denied, must be superuser to drop a directed query
ERROR 6905: Permission denied, must be superuser to save a query
ERROR 6906: Permission denied, must be superuser to update directed query status
ERROR 7376: Cannot close other user's sessions
ERROR 7463: Must be superuser to create notifier
ERROR 7464: Must be superuser to drop notifier
ERROR 7998: Permission denied for model string
ERROR 7999: Permission denied for schema string
ERROR 8569: Can't create the file [string]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42601

This topic lists the errors associated with the SQLSTATE 42601.

SQLSTATE 42601 Description

ERRCODE_SYNTAX_ERROR

Error messages associated with this SQLState

ERROR 2030: string has been deprecated as string string Vertica option
ERROR 2069: 'string' is not a table name in the current search_path
ERROR 2085: A column cannot occur in an equality predicate and an interpolation predicate
ERROR 2086: A column definition list is only allowed for functions that return "record"
ERROR 2087: A column definition list is required for functions returning "record"
ERROR 2093: A join can have only one set of interpolated predicates
ERROR 2100: A query with Time Series Aggregate Function string must have a timeseries clause
ERROR 2156: All columns are evaluated by expressions. At least one column should be read from input
ERROR 2157: All columns in select list must be columns used by projection
ERROR 2164: Alter Column Type driver: Unrecognized command type
ERROR 2180: Analytic function string must have an OVER clause
ERROR 2187: Analytic functions cannot be nested
ERROR 2191: ANALYZE CONSTRAINT is not supported
ERROR 2203: Anchor table not found
ERROR 2214: Argument value has invalid type value in ANALYZE_WORKLOAD
ERROR 2215: Argument value in ANALYZE_WORKLOAD must be constant
ERROR 2223: Argument in ANALYZE_CONSTRAINTS must be constant
ERROR 2230: Arguments of row IN must all be row expressions
ERROR 2238: At least two arguments are required
ERROR 2239: At most one path number can be entered
ERROR 2346: Cannot alter a sequence with START
ERROR 2374: Cannot compare row expressions of zero length
ERROR 2381: Cannot create a sequence with RESTART
ERROR 2444: Cannot insert into or update IDENTITY/AUTO_INCREMENT column "string"
ERROR 2445: Cannot insert into system column "string"
ERROR 2446: Cannot insert multiple commands into a prepared statement
ERROR 2521: Cannot specify anything other than user defined transforms string in the string list
ERROR 2525: Cannot specify more than one user-defined transform function in the SELECT list
ERROR 2526: Cannot specify more than one window clause with a user defined transform
ERROR 2534: Cannot use "PR" with "S"/"PL"/"MI"/"SG"
ERROR 2535: Cannot use "S" with "MI"
ERROR 2536: Cannot use "S" with "PL"
ERROR 2537: Cannot use "S" with "PL"/"MI"/"SG"/"PR"
ERROR 2538: Cannot use "S" with "SG"
ERROR 2539: Cannot use "V" with a decimal point
ERROR 2545: Cannot use aggregate function in VALUES
ERROR 2627: Column "string" in ENCODED BY clause is not found in the table
ERROR 2641: Column "string.string" must appear in the PARTITION BY list of Timeseries clause or be used in a Time Series Aggregate Function
ERROR 2642: Column string cannot be evaluated
ERROR 2645: Column string has other computed columns in its expression
ERROR 2647: Column string in ORDER BY list is not found in TABLE
ERROR 2659: Column alias list for "string" has too many entries
ERROR 2669: COLUMN OPTION is not supported
ERROR 2670: Column options are not supported
ERROR 2696: Conflicting INTERVAL subtypes
ERROR 2697: Conflicting NULL/NOT NULL declarations for column "string" of table "string"
ERROR 2715: Constraint declared INITIALLY DEFERRED must be DEFERRABLE
ERROR 2754: COPY requires a data source; either a FROM clause or a WITH SOURCE for a user-defined source
ERROR 2764: COPY: Expression for column string cannot be coerced
ERROR 2946: CREATE TABLE AS specifies too many column names
ERROR 2947: CREATE VIEW specifies more column names than columns
ERROR 2986: Database name is required (too few dotted names): string
ERROR 3023: Default values specified for IDENTITY/AUTO_INCREMENT column "string" of table "string"
ERROR 3125: Drop Column driver: Unrecognized command type
ERROR 3142: Duplicate column "string" in create table statement
ERROR 3143: Duplicate column string in constraint
ERROR 3146: Duplicate columns in select list of projection not allowed
ERROR 3151: Duplicate tables in projection not allowed
ERROR 3155: Duplicated parameters string not allowed
ERROR 3158: Each string query must have the same number of columns
ERROR 3164: Empty column name is invalid
ERROR 3165: Empty constraint name is invalid
ERROR 3171: ENCODED BY is not supported in CREATE PROJECTION statement when column renaming list is defined
ERROR 3172: ENCODED BY is not supported in CREATE PROJECTION statement with column definition list
ERROR 3173: ENCODED BY is not supported in CREATE TABLE AS SELECT statement when column list is defined
ERROR 3176: End epoch (value) number out of range
ERROR 3177: End epoch (value) precedes start epoch (value)
ERROR 3183: Epoch number out of range
ERROR 3185: Epoch time out of range
ERROR 3344: EXPORT ... SELECT may not specify INTO
ERROR 3348: Expression "<(string) - (string)> (interval qualifier)" is not supported
ERROR 3349: Expression for column string cannot be coerced
ERROR 3458: Function string is not allowed in Time Series queries
ERROR 3461: Function string requires at least one argument
ERROR 3487: Group by is not allowed in a projection
ERROR 3500: HAVING / GROUP BY not allowed with Time Series query
ERROR 3511: IGNORE NULLS can only be used with FIRST_VALUE or LAST_VALUE or NTH_VALUE
ERROR 3517: Improper %TYPE reference (too few dotted names): string
ERROR 3518: Improper %TYPE reference (too many dotted names): string
ERROR 3519: Improper qualified column name: string
ERROR 3520: Improper qualified name (too many dot): string
ERROR 3521: Improper qualified name (too many dots): string
ERROR 3522: Improper qualified name (too many dotted names): string
ERROR 3523: Improper relation name (too many dotted names): string
ERROR 3538: Incorrect parameter type provided: string is supposed to be of type string
ERROR 3548: Indirection is not allowed in a target column
ERROR 3549: Indirection is not allowed in the name of a FILLER column
ERROR 3571: INSERT ... SELECT may not specify INTO
ERROR 3572: INSERT has more expressions than target columns
ERROR 3573: INSERT has more target columns than expressions
ERROR 3599: Interpolated predicates are allowed only in ON CLAUSE of ANSI Join syntax
ERROR 3602: Interpolated predicates should refer to columns from both relations of the join
ERROR 3615: INTO is only allowed on first SELECT of UNION/INTERSECT/EXCEPT
ERROR 3619: Invalid argument type value in ANALYZE_CONSTRAINTS
ERROR 3672: Invalid hexadecimal number at or near "string"
ERROR 3706: Invalid node name in hint
ERROR 3709: Invalid number at or near "string"
ERROR 3738: Invalid projection name in hint: string
ERROR 3775: Invalid Unicode escape character 'character'
ERROR 3776: Invalid Unicode hex number "string"
ERROR 3812: Join condition in merge query must include at least one table attribute
ERROR 3841: Label can accept only one argument
ERROR 3865: LIMIT #,# syntax is not supported
ERROR 3944: Misplaced DEFERRABLE clause
ERROR 3945: Misplaced INITIALLY DEFERRED clause
ERROR 3946: Misplaced INITIALLY IMMEDIATE clause
ERROR 3947: Misplaced NOT DEFERRABLE clause
ERROR 3949: Missing argument
ERROR 3958: Missing savepoint name
ERROR 3959: Missing the path number
ERROR 3976: Multiple assignments to same column "string"
ERROR 3978: Multiple decimal points
ERROR 3979: Multiple default values specified for column "string" of table "string"
ERROR 3980: Multiple DEFERRABLE/NOT DEFERRABLE clauses not allowed
ERROR 3981: Multiple FOR UPDATE clauses are not allowed
ERROR 3982: Multiple INITIALLY IMMEDIATE/DEFERRED clauses not allowed
ERROR 3984: Multiple LIMIT clauses are not allowed
ERROR 3985: Multiple OFFSET clauses are not allowed
ERROR 3986: Multiple ORDER BY clauses are not allowed
ERROR 4023: Must specify memorySize parameter
ERROR 4024: Must specify one new name for each schema
ERROR 4025: Must specify one new name for each table
ERROR 4026: Must specify one new name for each view
ERROR 4062: NEW used in query that is not in a rule
ERROR 4066: No actions specified
ERROR 4070: No columns specified in select list
ERROR 4072: No constraints defined
ERROR 4105: No second argument needed when analyzing all constraints
ERROR 4136: Node "string" does not exist
ERROR 4161: Non-integer constant in string
ERROR 4164: Nonexistent columns: 'string'
ERROR 4203: Number of columns defined in CREATE TABLE statement is less than in SELECT query output
ERROR 4204: Number of columns defined in CREATE TABLE statement is more than in SELECT query output
ERROR 4285: Number of columns in the PROJECTION statement must be the same as the number of columns in the SELECT statement
ERROR 4225: OLD used in query that is not in a rule
ERROR 4227: ON COMMIT clause may only be specified for TEMPORARY tables
ERROR 4237: Only a single "S" is allowed
ERROR 4239: Only ASC is allowed in ORDER BY list of auto projection for CREATE TABLE
ERROR 4240: Only columns are allowed in ORDER BY list of auto projection for CREATE TABLE
ERROR 4241: Only columns are allowed in SELECT list of projection
ERROR 4247: Only inner joins are allowed in a projection defining query
ERROR 4268: Only tables are allowed in FROM clause of projection
ERROR 4291: Operator too long at or near "string"
ERROR 4294: Option string conflicts with prior options
ERROR 4296: Options not set
ERROR 4297: ORDER BY column in timeseries OVER clause must be Timestamp type
ERROR 4325: Parameters can only contain constants or constant expressions
ERROR 4327: Parsing error "string" at or near "string"
ERROR 4348: Path Number must be in [ 0, value ]
ERROR 4350: Pattern "0" must come before "PR"
ERROR 4351: Pattern "9" must come before "PR"
ERROR 4383: plannedConcurrency must be greater than 0
ERROR 4487: Projections can only be sorted in ascending order
ERROR 4629: Row expressions being compared must have the same number of entries
ERROR 4669: SELECT * with no tables specified is not valid
ERROR 4670: SELECT DISTINCT ON is not standard SQL, use just SELECT DISTINCT
ERROR 4706: Sequence functions accept constant strings arguments only
ERROR 4707: Sequence Manipulation functions are allowed in OUTER SELECT LIST only and cannot be in SELECT LIST of a WITH clause
ERROR 4732: Set Operators are not allowed in a projection
ERROR 4761: Sort key string should be in the target list
ERROR 4828: Subquery has too few columns
ERROR 4829: Subquery has too many columns
ERROR 4831: Subquery in FROM may not have SELECT INTO
ERROR 4833: Subquery in FROM must have an alias
ERROR 4835: Subquery must return a column
ERROR 4836: Subquery must return only one column
ERROR 4837: Subquery not allowed in a projection
ERROR 4838: Subquery not allowed in SELECT list and/or ORDER BY clause for Time Series queries
ERROR 4855: Syntactic Optimizer requires joins written using ANSI JOIN syntax
ERROR 4856: Syntax error at or near "string"
ERROR 4947: The foreign key in this constraint has already been defined as a foreign key for relation "string"
ERROR 4955: The number of target columns (value) does not match the number of columns (value) in the EXPORT statement
ERROR 4956: The number of target columns (value) is less than the number of columns (value) in the EXPORT statement
ERROR 5007: Time Series Aggregate Functions cannot be nested
ERROR 5008: Time Series queries cannot refer to column of outer query
ERROR 5009: Time Series queries cannot refer to column of outer query: "string.string"
ERROR 5011: Time slice length must be a positive integer constant
ERROR 5012: Time slice length must be an interval constant
ERROR 5161: Unequal number of entries in row expression
ERROR 5162: Unequal number of entries in row expressions
ERROR 5272: Unsupported From clause expression
ERROR 5285: Unsupported SET option
ERROR 5286: Unsupported SET option string
ERROR 5287: Unsupported SHOW option string
ERROR 5290: Unsupported transaction option string
ERROR 5305: Unterminated /* comment at or near "string"
ERROR 5306: Unterminated bit string literal at or near "string"
ERROR 5307: Unterminated dollar-quoted string at or near "string"
ERROR 5308: Unterminated hexadecimal string literal at or near "string"
ERROR 5310: Unterminated quoted identifier at or near "string"
ERROR 5311: Unterminated quoted string at or near "string"
ERROR 5323: Usage: clear_profiling( string , string )
ERROR 5324: Usage: disable_profiling( string )
ERROR 5325: Usage: enable_profiling( string )
ERROR 5326: Use "string(*)" to call this aggregate function
ERROR 5383: User Defined Transform Functions are allowed only in a SELECT list
ERROR 5386: User defined transform will return value columns, whereas value aliases provided
ERROR 5401: User-defined transform function string must have an OVER clause
ERROR 5413: Value must be either "units" or "plain"
ERROR 5415: Value must be either ON or OFF
ERROR 5452: Virtual tables are not allowed in FROM clause of projection
ERROR 5492: Wrong number of parameters for prepared statement "string"
ERROR 5493: Wrong number of parameters on left side of OVERLAPS expression
ERROR 5494: Wrong number of parameters on right side of OVERLAPS expression
ERROR 5505: You can specify a node name only once in a create projection statement, node string appears more than once
ERROR 5518: Zero-length delimited identifier at or near "string"
ERROR 5524: A projection can have only one basename
ERROR 5525: A projection can have only one createtype
ERROR 5566: Dimension tables may not have data that shorter lived than the fact table
ERROR 5577: Expression for user-defined type column string cannot be coerced
ERROR 5600: Invalid predicate in projection-select. Only PK=FK equijoins are allowed
ERROR 5617: Multiple WITH clauses not allowed
ERROR 5629: Not a Star or Snow-Flake Query
ERROR 5630: Nullable FKs are not allowed in projection definition
ERROR 5651: Recursive With is not supported
ERROR 5664: Subqueries not allowed in projection definition
ERROR 5670: The number of alias columns must be the same as the number of selected columns
ERROR 5691: User-defined function string is not a supported scalar function
ERROR 5696: WITH query name "string" specified more than once
ERROR 5711: Invalid function arguments
ERROR 5714: Missing the random seed
ERROR 5730: The second argument, sampling method, should be always be 1 -- naive sampling(biased)
ERROR 5733: The third argument must be large than 0
ERROR 5734: Three arguments at most: sampling seed, sampling method (optional, default 1), sampling size (optional, default 10)
ERROR 5916: If specified, maximum error percentage must be a numeric constant
ERROR 5926: Internal error parsing function string
ERROR 5929: Invalid maximum error percentage specified
ERROR 6034: Syntax Error: 'string' is a built in type
ERROR 6048: The minimum value that may be specified for maximum error percentage is 0.88
ERROR 6061: Too many arguments to string
ERROR 6119: 'deleted' hint takes no arguments
ERROR 6120: 'latestdata' hint arguments must be strings
ERROR 6122: A projection can replace only one original
ERROR 6124: A single argument must be supplied to string
ERROR 6136: Aggregate projection without group-by columns is not supported
ERROR 6137: Aggregate projections must have at least one aggregate (SUM, COUNT, MIN, or MAX)
ERROR 6140: All sort keys should be in the target list of the projection
ERROR 6146: At least two columns must be specified
ERROR 6196: Columns in ORDER-BY must be defined in the SELECT statement
ERROR 6197: Columns/Expressions in GROUP BY/PARTITION BY must be first and in the same order with columns in SELECT
ERROR 6198: ORDER BY columns/expressions in the OVER() clause must be the first SELECT columns/expressions not specified by PARTITION BY clause, and must be specified in SELECT list order
ERROR 6199: Columns/Expressions in the PARTITION BY clause may not be repeated in the ORDER BY clause
ERROR 6232: DISTINCT Aggregates are not allowed in projection
ERROR 6293: Interpolated join predicates cannot have expressions or functions over join columns
ERROR 6294: Invalid Argument
ERROR 6321: Limit in Top-K query must be a positive number
ERROR 6324: Limit/Offset is only allowed in topk projections
ERROR 6340: Multiple ORDER BY clauses in partition TopK/Limit query are not allowed
ERROR 6341: Multiple PARTITION clauses in TopK/Limit query are not allowed
ERROR 6366: Only one table or projection is allowed in FROM clause of aggregate projection
ERROR 6367: Only one table or projection is allowed in FROM clause of top-k projection
ERROR 6369: Only SUM, MIN, MAX and COUNT are allowed in aggregate projections
ERROR 6370: Only table or non-prejoin projection is allowed in FROM clause of aggregate projection
ERROR 6371: Only table or non-prejoin projection is allowed in FROM clause of top-k projection
ERROR 6372: ORDER BY is not allowed in aggregate projection. The aggregate projection is automatically ordered on group by columns
ERROR 6373: ORDER BY is not allowed in top-k projections. The top-k projection is automatically ordered on partition by columns
ERROR 6379: PARTITION BEST can only be used with single-phase user defined transform functions
ERROR 6381: PARTITION NODES can only be used with single-phase user defined transform functions
ERROR 6394: SEGMENTED BY / UNSEGMENTED is not allowed in aggregate projection. The aggregate projection is automatically segmented on group by columns
ERROR 6395: SEGMENTED BY / UNSEGMENTED is not allowed in top-k projections. The top-k projection is automatically segmented on partition by columns
ERROR 6518: At most one projection offset number can be entered
ERROR 6539: User Defined Transform string returns a string on which collation can't be applied in non default locale
ERROR 6543: Grouping functions only allowed with aggregates
ERROR 6544: Multilevel Aggregates are not allowed in projection
ERROR 6562: Hint PROJS cannot be used with hint SKIP_PROJS for the same table instance. Hint PROJS will be ignored
ERROR 6563: Hint SKIP_PROJS cannot be used with hint PROJS for the same table instance. Hint SKIP_PROJS will be ignored
ERROR 6568: Invalid projection name in hint: string. The whole hint will be ignored
ERROR 6640: Argument of hint string must be a positive integer
ERROR 6654: Batch/Prepass may only be used in the over(...) clause of a user defined transform in a Live Aggregate Projection
ERROR 6686: Cannot specify LIMIT with OVER(...) clause with a user defined transform
ERROR 6696: Column "string.string" refers to table "string" which is out of scope
ERROR 6697: Columns/Expressions in the partition clause have to appear in the SELECT list
ERROR 6701: Constants with the same parameter hint must be identical
ERROR 6800: Hint string must have one argument
ERROR 6834: Invalid use of _oidrefs hint
ERROR 6853: Live aggregate projections may only be one of the following types: Group-by, Top-K, or UDT
ERROR 6878: No previously saved query to associate with
ERROR 6899: Parameters cannot have NULL values
ERROR 6969: Subqueries for aggregate projections need to contain User Defined Transforms
ERROR 6972: Syntax Error. Struct member 'string' has missing base type
ERROR 7014: UD Parameters value length [value] is more than the max allowed length [value]
ERROR 7051: Unsupported hint for input expr
ERROR 7109: Parameter 'string' is required
ERROR 7147: Option string is repeated. Please specify each option only once
ERROR 7153: Syntax Error. At least one parameter has to be specified
ERROR 7164: Cannot use ORC format with FLEX tables
ERROR 7228: Cannot use string compression with string files
ERROR 7229: Cannot use PARQUET format with FLEX tables
ERROR 7327: SET USING cannot be specified for IDENTITY/AUTO_INCREMENT column "string" of table "string"
ERROR 7391: Cannot use an UDSource with string files
ERROR 7420: Expression list has more than one string column
ERROR 7488: Number of key/value pairs [value] exceeds the maximum allowed
ERROR 7538: Sequence manipulation functions not supported in INSERT SELECT with implicit column that has default query
ERROR 7744: Try local hint only supports SELECT queries without table join
ERROR 7746: Try local hint requires all nodes to be up
ERROR 7750: WHERE clause syntax is incompatible with "try local" hint, hint is ignored
ERROR 7785: SET USING/DEFAULT cannot be specified for IDENTITY/AUTO_INCREMENT column "string" of table "string"
ERROR 7909: Columns/Expressions in PARTITION BY must be first and in the same order with columns in SELECT
ERROR 7943: Cannot modify column with SET USING expression "string"
ERROR 7965: At most one WHEN MATCHED clause is allowed in MERGE
ERROR 7966: At most one WHEN NOT MATCHED clause is allowed in MERGE
ERROR 7967: MERGE/INSERT has more expressions than target columns
ERROR 7968: MERGE/INSERT has more target columns than expressions
ERROR 7982: Cannot use expressions in OVER(...) clause
ERROR 7989: Invalid column in MERGE statement: string. The insert filter of a WHEN NOT MATCHED clause cannot reference the target table string
ERROR 8011: SELECT expressions must have column labels
ERROR 8114: Multiple SET USING values specified for column "string" of table "string"
ERROR 8116: No '=' found in 'string', while parsing for string
ERROR 8164: Subquery in MERGE statement are allowed at SOURCE only
ERROR 8205: Hint string cannot be used on NULL

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42602

This topic lists the errors associated with the SQLSTATE 42602.
SQLSTATE 42602 Description

ERRCODE_INVALID_NAME

Error messages associated with this SQLState

ERROR 2383: Cannot create projections due to naming conflicts with existing projections
ERROR 2398: Cannot determine the best encoding options for some columns in table string.string due to insufficient data
ERROR 3059: DEPRECATED syntax. Segment expression "string" is a projection column name, segmenting on attribute "string"stringstringstring instead
ERROR 3378: Failed to generate a unique relation or sequence name
ERROR 3674: Failed to generate a unique relation or sequence name
ERROR 3703: Invalid name syntax
ERROR 3747: Invalid savepoint identifier string
ERROR 4159: Non-ASCII characters in names are prohibited
ERROR 4267: Only table column names & filler column names can appear in the list
ERROR 4451: Projection "string" does not exist
ERROR 4506: Query weight must be positive
ERROR 5360: User "string" does not exist
ERROR 5403: User/role "string" already exists
ERROR 5569: Either column "string" does not exist or table alias "string" is not allowed in "WHEN MATCHED THEN UPDATE SET"
ERROR 5769: Cannot drop the main vertica license
ERROR 5968: No such license string to drop
ERROR 5970: Node string is not a control node
ERROR 6089: Unknown control node string
ERROR 6676: Cannot drop global heir user "string". Try changing / clearing the GlobalHeirUsername config parameter
ERROR 7117: Refresh table name error: string
ERROR 7893: Invalid model name
ERROR 8304: Parameter has the same name as argument: string
ERROR 8403: Cannot drop the only AutoPass license

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42611

This topic lists the errors associated with the SQLSTATE 42611.

SQLSTATE 42611 Description
ERRCODE_INVALID_COLUMN_DEFINITION

Error messages associated with this SQLState

ERROR 6099: Using LONG column 'string' in a constraint
ERROR 6100: Using PARTITION expression that returns a string value
ERROR 6101: Using PARTITION expression that returns a LONG value
ERROR 6102: Using PARTITION expression that returns a LONG value: 'string'
ERROR 7342: string expression for column "string" may not refer to itself
ERROR 7389: Cannot set string for column "string" since it is referenced in default expression of column "string"
ERROR 7783: Cannot set string for column "string" since it is referenced in string expression of column "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42622

This topic lists the errors associated with the SQLSTATE 42622.

SQLSTATE 42622 Description
ERRCODE_NAME_TOO_LONG
Error messages associated with this SQLState

ERROR 2462: Cannot open FileColumn because path is too long string
ERROR 3507: Identifier "string" is value octets long. Maximum limit is value octets

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42701

This topic lists the errors associated with the SQLSTATE 42701.

SQLSTATE 42701 Description

ERRCODE_DUPLICATE_COLUMN

Error messages associated with this SQLState

ERROR 2629: Column "string" is already of type "string"
ERROR 2638: Column "string" specified more than once
ERROR 2654: Column string specified more than once
ERROR 2655: Column string specified more than once in options list
ERROR 2662: Column name "string" already exists
ERROR 2663: Column name "string" appears more than once in USING clause
ERROR 2664: Column name "string" does not exist
ERROR 3144: Duplicate column string in ORDER BY list
ERROR 3145: Duplicate column name
ERROR 3150: Duplicate projection column name (projection: string)
ERROR 3154: Duplicated parameter "string" in parameter list
ERROR 5450: View definition can not contain duplicate column names "string"
ERROR 5878: Failed to create table string: duplicate column name string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42702

This topic lists the errors associated with the SQLSTATE 42702.

SQLSTATE 42702 Description

ERRCODE_AMBIGUOUS_COLUMN

Error messages associated with this SQLState

ERROR 2604: Clause string "string" is ambiguous
ERROR 2671: Column reference "string" is ambiguous
ERROR 2681: Common column name "string" appears more than once in left table
ERROR 2682: Common column name "string" appears more than once in right table
ERROR 5904: Flex table "string" has no internal "string" column
ERROR 6144: Ambiguous column reference in projection

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42703

This topic lists the errors associated with the SQLSTATE 42703.

SQLSTATE 42703 Description

ERRCODE_UNDEFINED_COLUMN
Error messages associated with this SQLState

ERROR 2359: Cannot assign to field "string" of column "string" because there is no such column in data type string
ERROR 2625: Column "string" does not exist;  
Vertica does not support 'SELECT <table_name> FROM <table_name>'
ERROR 2633: Column "string" named as primary key does not exist
ERROR 2634: Column "string" not found in data type string
ERROR 2635: Column "string" of relation "string" does not exist
ERROR 2636: Column "string" specified in USING clause does not exist in left table
ERROR 2637: Column "string" specified in USING clause does not exist in right table
ERROR 2639: Column "string"."string" does not exist as a projection column
ERROR 2643: Column string does not exist
ERROR 2644: Column string does not exist in table
ERROR 2651: Column string must be loaded or computed
ERROR 2656: Column string.string does not exist
ERROR 2870: Could not identify column "string" in record data type
ERROR 4450: Projection "string" does not contain column "string"
ERROR 6695: Column "string"."string" must have CHAR, VARCHAR, LONG VARCHAR, VARBINARY, LONG VARBINARY, or USER DEFINED type
ERROR 7009: Tokenizer must have a "string" column output for stemming
ERROR 7167: Column "string" does not exist on table "string"
ERROR 7367: After excluding columns, no input column remains
ERROR 7959: Column [string] does not exist in relation [string]
ERROR 7984: Column [string] is duplicated in parameter string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42704

This topic lists the errors associated with the SQLSTATE 42704.

SQLSTATE 42704 Description

ERRCODE_UNDEFINED_OBJECT
Error messages associated with this SQLState

ERROR 2067: 'string' is not a known granularity for audits.
  string
ERROR 2068: 'string' is not a known TM task.
  string
ERROR 2070: 'string' is not a valid granularity for string.
  string
ERROR 2073: 'string' is not supported by index tool
ERROR 2274: Bootstrap error (most likely in Bootstrap.cpp): Unregistered name string
ERROR 2275: Bootstrap error (most likely in Bootstrap.cpp): Unregistered oid value
ERROR 2710: Constraint "string" does not exist
ERROR 2711: Constraint "string" does not exist on table "string"
ERROR 3001: DDL statement interfered with string.nextval
ERROR 3256: Error reported by client: string
ERROR 3442: Found eligible value processes to invite, but no matching nodes in catalog
ERROR 3637: Invalid Component Name 'string'
ERROR 3655: Invalid epoch range
ERROR 3698: Invalid mergeout task identifier (Possible values are: [0, value])
ERROR 3715: Invalid object name
ERROR 3748: Invalid scope in ANALYZE_WORKLOADstring: schema or table string was altered
ERROR 3749: Invalid scope in ANALYZE_WORKLOAD: schema or table string does not exist
ERROR 3756: Invalid Sub-Component Name 'string'
ERROR 3769: Invalid TM operation
ERROR 3779: Invalid user ID: value
ERROR 3842: Language does not exist: string
ERROR 3855: Library "string" does not exist
ERROR 3862: Library with name 'string' does not exist
ERROR 4046: Network Interface "string" does not exist
ERROR 4047: Network Interface "string" is setup on another node
ERROR 4101: No role "string" exists
ERROR 4109: No storages in the specified epoch range
ERROR 4110: No such node string
ERROR 4111: No such object
ERROR 4112: No such projection
ERROR 4113: No such projection 'string'
ERROR 4123: No user or role "string" exists
ERROR 4129: No value found for parameter "string"
ERROR 4130: No value found for parameter value
ERROR 4137: Node string does not exist
ERROR 4216: Object 'string' is not a projection
ERROR 4217: Object 'string' is not a table
ERROR 4218: Object 'string' is not a table or projection
ERROR 4223: OID value is not a sequence
ERROR 4224: OID value is not a Table or a View
ERROR 4446: Profile "string" does not exist
ERROR 4447: Profile 'string' does not exist
ERROR 4594: Resource pool "string" does not exist
ERROR 4596: Resource pool 'string' does not exist
ERROR 4614: Role "string" does not exist
ERROR 4616: Role "string" not found
ERROR 4650: Schema "string" does not exist
ERROR 4656: Schema, table, or projection "string" does not exist.
  string
ERROR 4697: Sequence "string" does not exist
ERROR 4713: Sequence with name 'string' does not exist
ERROR 4806: Subnet "string" does not exist
ERROR 4928: The string ["string"] does not exist
ERROR 5105: Tuning rule "string" does not exist
ERROR 5108: Type "string" does not exist
ERROR 5109: Type "string" is only a shell
ERROR 5112: Type string is only a shell
ERROR 5115: Type with OID value does not exist
ERROR 5227: Unrecognized drop object type: value
ERROR 5362: User or Role "string" not found
ERROR 5365: User available location ["string"] does not exist on node ["string"]
ERROR 5446: View "string" does not exist
ERROR 5459: Window "string" does not exist
ERROR 5532: Can not find any eligible locations in tier string
ERROR 5585: Fault Group "string" does not exist
ERROR 5614: Library string does not exist
ERROR 5688: User Defined Type "string" does not exist
ERROR 5797: Could not find the JVM resource pool
ERROR 5913: HCatalog database string does not exist
ERROR 5931: Invalid Policy Name 'string'
ERROR 5965: New node cannot be placed in a non-existent Fault Group "string"
ERROR 5967: No database found. Create/use a database before start using DFS
ERROR 5969: No table or projection named string exists
ERROR 5974: Node doesn't exist
ERROR 5977: Object does not exist
ERROR 6071: Type value with odbc_subtype value is not supported
ERROR 6149: Authentication "string" does not exist
ERROR 6156: Can not use empty authentication
ERROR 6285: Index "string" does not exist
ERROR 6354: No auth "string" exists
ERROR 6358: No such table 'string'
ERROR 6423: Target standby node "string" does not exist
ERROR 6567: Invalid input parameters.
        string
ERROR 6887: OID value is of a type unsupported for re-parenting
ERROR 7101: Invalid role oid encountered
ERROR 7275: Registry::getSequenceCache failed to get Sequence Cache in planning phase for [ value ]
ERROR 7704: Blob does not exist
ERROR 7897: Model "string" does not exist
ERROR 7901: Model with name 'string' does not exist
ERROR 8112: Model [string] does not exist or access was denied
ERROR 8163: Source node string is not subscribed to shard string
ERROR 8166: Subscription identified by (string,string) does not exist
ERROR 8174: The model does not exist
ERROR 8248: Cannot make a string subscription as PRIMARY
ERROR 8252: Subscription of node string to shard string is already PRIMARY
ERROR 8282: Invalid DFS Root Type
ERROR 8287: No Model found. Create a model before start using DFS
ERROR 8288: Trouble finding model files
ERROR 8324: Subscription doesn't exist or not in PASSIVE state
ERROR 8560: Model [string] has unexpectedly changed
ERROR 8584: Model "value" does not exist
ERROR 8591: Source Subscription value does not exist. Abort and retry..

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42710

This topic lists the errors associated with the SQLSTATE 42710.

SQLSTATE 42710 Description

ERRCODE_DUPLICATE_OBJECT
Error messages associated with this SQLState

ERROR 2101: A sequence named "string" already exists
ERROR 2105: A table named "string" already exists
ERROR 2107: A view named "string" already exists
ERROR 2273: Bootstrap error (most likely in Bootstrap.cpp): Oid value is already registered
ERROR 2276: Bootstrap error (most likely in Bootstrap.cpp): Name string is already registered
ERROR 2713: Constraint string already exists
ERROR 3153: Duplicated local temp table found in design queries: string
ERROR 3327: Existing object "string" is not a view
ERROR 3881: Location [string] already exists for node string
ERROR 4043: Network Interface "string" already exists
ERROR 4135: Node "string" already exists
ERROR 4213: Object "string" already exists
ERROR 4445: Profile "string" already exists
ERROR 4482: Projection with base name "string" already exists
ERROR 4564: Relation "string" already exists
ERROR 4565: Relation "string" already exists in schema "string"
ERROR 4593: Resource pool "string" already exists
ERROR 4621: Role "string" already exists
ERROR 4804: Subnet "string" already exists
ERROR 4805: Subnet "string" already exists for [string]
ERROR 5582: Fault Group "string" already exists
ERROR 5584: Fault Group "string" cannot depend on itself directly or indirectly
ERROR 5615: Location [string] conflicts with existing location [string] on node string
ERROR 5623: Network Interface "string" already exists for [string]
ERROR 6089: Resource pool string already exists
ERROR 6112: Unable to guarantee the same base name for all segmented buddy projections
ERROR 6148: Authentication "string" already exists
ERROR 6157: Can’t create an index in schema "string": Schema cannot be found
ERROR 6158: Can’t create an index named "string": Object already exists
ERROR 6188: Client authentication "string" already exists
ERROR 6735: Directed query with identifier "string" already exists
ERROR 6736: Directed query with identifier "string" does not exist
ERROR 7059: User "string" already exists
ERROR 7713: Error in blob creation: A blob with name 'string' already exists
ERROR 7963: Relation [string] specified in parameter 'output_view' already exists
ERROR 8059: Cannot add more than one cache location
ERROR 8201: The specified output Table/View [string] already exists
ERROR 8251: Node string is already subscribed to shard string in string state
ERROR 8398: A model named "string" already exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42712

This topic lists the errors associated with the SQLSTATE 42712.

SQLSTATE 42712 Description

ERRCODE_DUPLICATE_ALIAS

Error messages associated with this SQLState

ERROR 4901: Table name "string" specified more than once

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42723

This topic lists the errors associated with the SQLSTATE 42723.

SQLSTATE 42723 Description

ERRCODE_DUPLICATE_FUNCTION

Error messages associated with this SQLState

ERROR 2278: Built-in function with the same name already exists: string
Error Messages Associated with SQLSTATE 42725

This topic lists the errors associated with the SQLSTATE 42725.

SQLSTATE 42725 Description

ERRCODE_AMBIGUOUS_FUNCTION

Error messages associated with this SQLState

ERROR 3459: Function string is not unique
ERROR 4289: Operator is not unique: string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42803

This topic lists the errors associated with the SQLSTATE 42803.

SQLSTATE 42803 Description

ERRCODE_GROUPING_ERROR

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error messages associated with this SQLState

ERROR 2134: Aggregate function calls in subqueries cannot refer to columns in parent (outer) query
ERROR 2135: Aggregate function calls may not be nested
ERROR 2140: Aggregates not allowed in GROUP BY clause
ERROR 2141: Aggregates not allowed in JOIN conditions
ERROR 2142: Aggregates not allowed in WHERE clause
ERROR 2219: Argument string must not contain aggregates
ERROR 2543: Cannot use aggregate function in EXECUTE parameter
ERROR 2544: Cannot use aggregate function in function expression in FROM
ERROR 2640: Column "string.string" must appear in the GROUP BY clause or be used in an aggregate function
ERROR 4300: ORDER/GROUP BY expression not found in targetlist
ERROR 4634: Rule WHERE condition may not contain aggregate functions
ERROR 4667: SEGMENTED BY expression may not contain aggregate functions
ERROR 4841: Subquery uses ungrouped column "string.string" from outer query
ERROR 6139: All non-aggregate expressions in the projection SELECT list must appear in the GROUP BY clause
ERROR 6187: Cannot use aggregate expressions in the projection SELECT list without a GROUP BY clause
ERROR 6277: Grouping functions not allowed in GROUP BY clause
ERROR 6278: Grouping functions not allowed in JOIN conditions
ERROR 6279: Grouping functions not allowed in WHERE clause
ERROR 7085: Aggregate function calls cannot contain grouping functions
ERROR 7099: Grouping functions cannot be nested
ERROR 7182: Grouping function arguments need to be group by expressions

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42804

This topic lists the errors associated with the SQLSTATE 42804.

SQLSTATE 42804 Description

ERRCODE_DATATYPE_MISMATCH
Error messages associated with this SQLState

ERROR 2217: Argument string must be type float, not type string
ERROR 2218: Argument string must be type integer, not type string
ERROR 2222: Argument string must not return a set
ERROR 2224: Argument of string must be type boolean, not type string
ERROR 2225: Argument of string must not return a set
ERROR 2231: Array assignment requires type string but expression is of type string
ERROR 2232: Array assignment to "string" requires type string but expression is of type string
ERROR 2234: Array subscript must have type integer
ERROR 2358: Cannot assign to field "string" of column "string" because its type string is not a composite type
ERROR 2527: Cannot subscript type string because it is not an array
ERROR 2630: Column "string" is of type string but default expression is of type string
ERROR 2631: Column "string" is of type string but expression is of type string
ERROR 2846: Could not determine actual result type for function "string" declared to return type string
ERROR 2850: Could not determine row description for function returning record
ERROR 3429: For 'string', types string and string are inconsistent
ERROR 3447: Function "string" in FROM has unsupported return type string
ERROR 3545: Index expression may not return a set
ERROR 3801: IS DISTINCT FROM requires = operator to yield boolean
ERROR 3943: Mismatched types in VALUES LESS THAN expressions
ERROR 4069: No column alias was provided
ERROR 4199: Number of aliases does not match number of columns
ERROR 4284: Operator string must not return a set
ERROR 4285: Operator string must return type boolean, not type string
ERROR 4317: Parameter $value of type string cannot be coerced to the expected type string
ERROR 4625: Row comparison operator must not return a set
ERROR 4626: Row comparison operator must yield type boolean, not type string
ERROR 4883: Subfield "string" is of type string but expression is of type string
ERROR 6480: Column "string" is of type string but expression in Access Policy is of type string. It will be coerced at execution time
ERROR 7221: Argument of type string must be type boolean, not string
ERROR 7222: Argument of type string must be type boolean, not type string
ERROR 7309: Column "string" is of type string but set using expression is of type string
ERROR 7708: Column must be a string type
ERROR 7929: Column to be added/refreshed is of type string but default/set-using expression is of type string
Error Messages Associated with SQLSTATE 42809

This topic lists the errors associated with the SQLSTATE 42809.

SQLSTATE 42809 Description

ERRCODE_WRONG_OBJECT_TYPE

Error messages associated with this SQLState

ERROR 2037: string is not a supported analytic function
ERROR 2131: Aggregate function calls cannot contain analytic function calls
ERROR 2132: Aggregate function calls cannot contain sequence function calls
ERROR 2668: Column notation .string applied to type string, which is not a composite type
ERROR 2755: COPY requires relation string to be a Table, not a string
ERROR 2810: Could not add location [string]: Directory not empty
ERROR 2811: Could not add location [string]: Not a directory
ERROR 3114: DISTINCT specified, but string is not an aggregate function
ERROR 3421: First argument to modularhash_wrapper must be an integer constant
ERROR 3422: First argument to modularhash_wrapper must be of type integer, not string
ERROR 3463: Function string(string) is not an aggregate
ERROR 3552: Inherited relation "string" is not a table
ERROR 3669: Invalid function given
ERROR 3965: modularhash_wrapper must have two arguments: an integer constant and a call to modularhash_internal
ERROR 3966: modularhash_wrapper second argument is not modularhash_internal or a constant
ERROR 4215: Object "string" is not a projection
ERROR 4270: Op ANY/ALL (array) requires array on right side
ERROR 4271: Op ANY/ALL (array) requires operator not to return a set
ERROR 4272: Op ANY/ALL (array) requires operator to yield boolean
ERROR 4542: Record type has not been registered
ERROR 4657: Second argument to string must be a non-negative integer constant
ERROR 4931: The argument to string cannot be null
ERROR 4932: The argument to `string` must be a constant
ERROR 4987: Third argument to `string` must be a constant
ERROR 5111: Type `string` is not composite
ERROR 6036: Table `string` is not a flex table
ERROR 6691: CAST and `current_date/current_time` functions are not supported as analytic functions
ERROR 6951: Skip lazy projection creation for table `string.string` since the object referenced by the hint is either invalid or of the wrong type (expected `string`)

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

**Error Messages Associated with SQLSTATE 42830**

This topic lists the errors associated with the SQLSTATE 42830.

**SQLSTATE 42830 Description**

**ERRCODE_INVALID_FOREIGN_KEY**

Error messages associated with this SQLState

ERROR 3438: Foreign keys not specified
ERROR 3531: Incompatible data types between primary and foreign key columns: `fk: string, pk: string`
ERROR 4207: Number of primary and foreign keys must be the same

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

**Error Messages Associated with SQLSTATE 42846**

This topic lists the errors associated with the SQLSTATE 42846.
SQLSTATE 42846 Description

ERRCODE_CANNOT_COERC

Error messages associated with this SQLState

ERROR 2015: string could not convert type string to string
ERROR 2366: Cannot cast type string to string
ERROR 4986: Third argument of string could not be converted from type string to type string
ERROR 6510: Column "string" is of type string but expression in Access Policy is of type string. Cannot coerce expression
ERROR 7310: Column "string" is of type string but the string expression is of type string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42883

This topic lists the errors associated with the SQLSTATE 42883.

SQLSTATE 42883 Description

ERRCODE_UNDEFINE_FUNCTION

Error messages associated with this SQLState

ERROR 2126: Aggregate string(string) does not exist
ERROR 2127: Aggregate string(*) does not exist
ERROR 3456: Function string does not exist
ERROR 3457: Function string does not exist, or permission is denied for string
ERROR 3462: Function string with the specified arguments does not exist
ERROR 3930: Meta-function string cannot be used in COPY
ERROR 3931: Meta-function string cannot be used in INSERT
ERROR 3932: Meta-function string cannot be used in UPDATE
ERROR 3933: Meta-function string cannot be used with FROM
ERROR 3934: Meta-function ("string") can be used only in the Select clause
ERROR 4067: No binary input function available for type string
ERROR 4068: No binary output function available for type string
ERROR 4083: No input function available for type string
ERROR 4091: No output function available for type string
ERROR 4286: Operator does not exist: string
ERROR 4290: Operator requires run-time type coercion: string
ERROR 5394: User procedure call (value) is not supported with FROM
ERROR 5910: Function string with the specified type and arguments does not exist
ERROR 6333: Meta-functions or non-deterministic functions cannot be used in projection column expressions
ERROR 6809: Inappropriate usage of meta-function ("string")
ERROR 7323: Meta-functions cannot be used in string expressions
ERROR 7324: Meta-functions cannot be used in string query definitions
ERROR 7340: VOLATILE functions cannot be used in a string expression when adding a column
ERROR 7341: VOLATILE functions cannot be used in a string query when adding a column
ERROR 7589: VOLATILE functions cannot be used in a string expression
ERROR 7590: VOLATILE functions cannot be used in a string query
ERROR 7657: Function "string" does not exist
ERROR 7688: Calling metafunctions 'string' and 'string' within the same query is not supported
ERROR 7692: Multiple calls to metafunction 'string' within the same query are not supported
ERROR 7970: Meta-function string cannot be used in MERGE/UPDATE

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42939

This topic lists the errors associated with the SQLSTATE 42939.

SQLSTATE 42939 Description

ERRCODE_RESERVED_NAME
Error messages associated with this SQLState

ERROR 2297: Can not drop default profile
ERROR 2299: Can not rename default profile
ERROR 2418: Cannot drop role "string"
ERROR 2448: Cannot rename role string
ERROR 2489: Cannot rename system column epoch
ERROR 2665: Column name "string" is reserved
ERROR 2666: Column name string is reserved
ERROR 3778: Invalid use of reserved the column name "string"
ERROR 4030: Names starting with "v_" are reserved names
ERROR 4953: The name "string" is a reserved name
ERROR 4962: The prefix "sys_" is reserved for system tuning rule
ERROR 6195: Column name "string" is reserved in aggregate projections
ERROR 7441: Invalid use of reserved column name "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42P20

This topic lists the errors associated with the SQLSTATE 42P20.

SQLSTATE 42P20 Description

ERRCODE_WINDOWING_ERROR

Error messages associated with this SQLState

ERROR 2011: string cannot use the WITHIN GROUP clause
ERROR 2041: string may only have one sort expression in the WITHIN GROUP clause
ERROR 2043: string must contain an ORDER BY clause within its analytic clause
ERROR 2044: string must NOT contain an ORDER BY clause or WINDOWING clause within its analytic clause
ERROR 2045: string must NOT contain WINDOWING clause within its analytic clause
ERROR 2047: string only supports the Integer, Float, Interval and Numeric data types
ERROR 2182: Analytic functions are allowed only in a SELECT list and/or ORDER BY clause
ERROR 2185: Analytic functions are not supported in the PARTITION BY of an OVER clause
ERROR 2188: Analytic functions must have a FROM clause
ERROR 2189: Analytic functions not allowed in string
ERROR 2305: Can’t cast the window bound into Int
ERROR 2306: Can’t cast the window bound into the same data type of the ORDER BY column
ERROR 2465: Cannot override ORDER BY clause of window "string"
ERROR 2466: Cannot override PARTITION BY clause of window "string"
ERROR 2524: Cannot specify frame clause of window "string"
ERROR 3435: For range moving window, OrderBy expression must be one of Int, Float, Time, Timestamp, Interval, Date or Numeric
ERROR 3446: Frame clause not allowed without windowing order by
ERROR 3839: Keyword "ALL" is invalid in analytic functions
ERROR 4362: PERCENTILE_CONT/PERCENTILE_DISC must have the WITHIN GROUP clause
ERROR 4363: PERCENTILE_CONT/PERCENTILE_DISC must NOT contain an ORDER BY clause or WINDOWING clause within its analytic clause
ERROR 4811: Subqueries are not supported in the PARTITION BY of a timeseries OVER clause
ERROR 5086: Time Series Aggregate Functions are not supported in the PARTITION BY of a timeseries OVER clause
ERROR 5087: Time Series timestamp alias/Time Series Aggregate Functions not allowed in string
ERROR 5460: Window "string" is already defined
ERROR 5461: Window frame cannot end with PRECEDING if start is CURRENT ROW
ERROR 5462: Window frame cannot end with PRECEDING or CURRENT ROW if start is FOLLOWING
ERROR 5463: Window frame cannot end with UNBOUNDED PRECEDING
ERROR 5464: Window frame cannot start with UNBOUNDED FOLLOWING
ERROR 5466: Window frame logical offset must be a non-negative number to be consistent with the sort column type
ERROR 5467: Window frame logical offset must be an Interval (Day to Second or Year to Month) to be consistent with the sort column type
ERROR 5468: Window frame logical offset must be an Interval (Day to Second) to be consistent with the sort column type
ERROR 5469: Window frame logical offset must be an interval to be consistent with the sort column type
ERROR 5470: Window frame logical offset must be Int when the sort column type is Int
ERROR 5471: Window frame logical offset must be the same type as the sort column type (Interval Day to Second)
ERROR 5472: Window frame logical offset must be the same type as the sort column type (Interval Year to Month)
ERROR 5473: Window frame logical or physical offset must be a constant
ERROR 5474: Window frame logical or physical offset must be non-negative number or interval
ERROR 5475: Window frame physical offset must be non-negative number
ERROR 5477: Window ordering clause can only contain a single sort key if RANGE is used
ERROR 5478: Windowing not supported for User Defined Analytic functions
ERROR 6276: Grouping functions cannot appear within Analytic functions
ERROR 6900: Partition Prepass/Batch may only be used with Live Aggregate Projection
ERROR 7838: Windowing not supported for analytic usage of BOOL_XOR

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V01

This topic lists the errors associated with the SQLSTATE 42V01.

SQLSTATE 42V01 Description

ERRCODE_UNDEFINED_TABLE

Error messages associated with this SQLState

ERROR 2308: Can’t find anchor table
ERROR 2312: Can’t find table
ERROR 2313: Can’t find table "string"
ERROR 2714: Constraint string does not exist
ERROR 2948: CTAS: table "string" was dropped in another session (DDL interference)
ERROR 3642: Invalid CTAS query: string
ERROR 3760: Invalid table name
ERROR 3761: Invalid table name "string"
ERROR 3762: Invalid table name string
ERROR 3953: Missing FROM-clause entry for table "string"
ERROR 3954: Missing FROM-clause entry in subquery for table "string"
ERROR 4416: Primary table "string" does not exist
ERROR 4566: Relation "string" does not exist
ERROR 4567: Relation "string" in FOR UPDATE clause not found in FROM clause
ERROR 4568: Relation "string.string" does not exist
ERROR 4570: Relation with OID value does not exist
ERROR 4883: Table "string.string" does not exist
ERROR 4898: Table does not exist (oid=value)
ERROR 4911: Table with OID value does not exist
ERROR 4912: Table/View with name 'string' does not exist
Error Messages Associated with SQLSTATE 42V02

This topic lists the errors associated with the SQLSTATE 42V02.

SQLSTATE 42V02 Description

ERRCODE_UNDEFINED_PARAMETER

Error messages associated with this SQLState

ERROR 3638: Invalid configuration parameter string; aborting configuration change
ERROR 4321: Parameter value is not set
ERROR 4984: There is no parameter $value
ERROR 5202: Unknown configuration parameter
ERROR 6201: Configuration parameter string cannot be set at this level; aborting configuration change
ERROR 6511: Configuration parameter string cannot be set at this level; no value to return
ERROR 7357: A problem occurred during the execution of a detect_outliers iteration.
  Detail: string
ERROR 7363: A problem occurred during the execution of balance.
  Detail: string
ERROR 7891: init_method not allowed with argument initial_centers_table
ERROR 8017: Type mismatch of an optional argument: string must be a valid integer
ERROR 8018: Type mismatch of an optional argument: string must be a valid numeric
ERROR 8019: Type mismatch of an optional argument: string must be a valid string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
ERROR 8051: A problem occurred during the execution of computing approximate mad table.
   Detail: string
ERROR 8052: A problem occurred during the execution of computing approximate median table.
   Detail: string
ERROR 8053: A problem occurred during the execution of computing approximate meanAD table.
   Detail: string
ERROR 8054: A problem occurred during the execution of computing final mad table.
   Detail: string
ERROR 8289: Type mismatch of an optional argument: parentName must be a valid string
ERROR 8290: Type mismatch of an optional argument: parentType must be a valid string
ERROR 8316: Type mismatch of an optional argument: string must be a valid boolean

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V03

This topic lists the errors associated with the SQLSTATE 42V03.

SQLSTATE 42V03 Description

ERRCODE_DUPLICATE_CURSOR

Error messages associated with this SQLState

ERROR 2615: Closing existing cursor "string"
ERROR 2968: Cursor "string" already exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42V04

This topic lists the errors associated with the SQLSTATE 42V04.

SQLSTATE 42V04 Description

ERRCODE_DUPLICATE_DATABASE

Error messages associated with this SQLState

ERROR 2706: Connection to database [string] already exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V06

This topic lists the errors associated with the SQLSTATE 42V06.

SQLSTATE 42V06 Description

ERRCODE_DUPLICATE_SCHEMA

Error messages associated with this SQLState

ERROR 4649: Schema "string" already exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42V07

This topic lists the errors associated with the SQLSTATE 42V07.

SQLSTATE 42V07 Description

ERRCODE_DUPLICATE_TABLE

Error messages associated with this SQLState

ERROR 4753: Skip lazy projection creation since super projection for table string.string already exists
ERROR 6952: Skip lazy projection creation since a projection enforcing key constraint 'string' for table string.string already exists
ERROR 8050: A problem occurred during execution of Balance - the desired temp table name [string] already exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V08

This topic lists the errors associated with the SQLSTATE 42V08.

SQLSTATE 42V08 Description

ERRCODE_AMBIGUOUS_PARAMETER

Error messages associated with this SQLState

ERROR 2848: Could not determine data type of parameter $value
ERROR 3534: Inconsistent types deduced for parameter $value

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V09

This topic lists the errors associated with the SQLSTATE 42V09.

SQLSTATE 42V09 Description

ERRCODE_AMBIGUOUS_ALIAS

Error messages associated with this SQLState

ERROR 4908: Table reference "string" is ambiguous
ERROR 4909: Table reference value is ambiguous

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V10

This topic lists the errors associated with the SQLSTATE 42V10.

SQLSTATE 42V10 Description

ERRCODE_INVALID_COLUMN_REFERENCE
Error messages associated with this SQLState

ERROR 2046: string not allowed in string clause
ERROR 2050: string position value is not in select list
ERROR 2221: Argument string must not contain variables
ERROR 3467: Function expression in FROM may not refer to other relations of same query level
ERROR 3820: JOIN/ON clause refers to "string", which is not part of JOIN
ERROR 4832: Subquery in FROM may not refer to other relations of same query level
ERROR 4877: Table "string" has value columns available but value columns specified
ERROR 5057: Too many column aliases specified for function string
ERROR 5194: UNION/INTERSECT/EXCEPT member statement may not refer to other relations of same query level

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V11

This topic lists the errors associated with the SQLSTATE 42V11.

SQLSTATE 42V11 Description

ERRCODE_INVALID_CURSOR_DEFINITION

Error messages associated with this SQLState

ERROR 2522: Cannot specify both SCROLL and NO SCROLL

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42V13

This topic lists the errors associated with the SQLSTATE 42V13.

SQLSTATE 42V13 Description

ERRCODE_INVALID_FUNCTION_DEFINITION

Error messages associated with this SQLState

ERROR 2038: string is not a supported Time Series Aggregate Function
ERROR 2139: Aggregates may not return sets
ERROR 2173: An error occurred on node [string] when setting up the function [string]: [string]
ERROR 2177: An error occurred when setting up function "string"
ERROR 2397: Cannot determine result data type
ERROR 2451: Cannot load data from 0 sources; please specify 1 or more (on node [string])
ERROR 2494: Cannot RETURNREJECTED with multiple files or data sources
ERROR 3113: DISTINCT is supported only for single-argument aggregates
ERROR 3476: Functions in language string can be created only in fenced mode
ERROR 3604: Interpolation scheme string for Time Series Aggregate Function string is not supported
ERROR 3708: Invalid null argument for TSA function string
ERROR 3843: Language(string) does not match the language associated with the library(string)
ERROR 3854: Length of a string in a return type must be greater than zero
ERROR 3860: Library file is not loaded
ERROR 3861: Library not found: string
ERROR 3929: Meta functions cannot be used in U Dx definitions
ERROR 4086: No language specified
ERROR 4095: No procedure source specified
ERROR 4096: No procedure user specified
ERROR 4251: Only one expression is allowed
ERROR 4257: Only simple "RETURN expression" is allowed
ERROR 4409: Precision of a numeric in a return type must be greater than zero
ERROR 4608: Return type string is not supported for SQL functions
ERROR 4609: Return type mismatch in a function declared to return string
ERROR 4610: Return type mismatch in function declared to return string
Error Messages Associated with SQLSTATE 42V15

This topic lists the errors associated with the SQLSTATE 42V15.

SQLSTATE 42V15 Description

ERRCODE_INVALID_SCHEMA_DEFINITION

Error messages associated with this SQLState

ERROR 2470: Cannot plan query because no super projections are safe, some node(s) are down
ERROR 2945: CREATE specifies a schema (string) different from the one being created (string)
ERROR 3365: Failed to create default projections for table "string"."string": string
ERROR 3586: Insufficient projections to answer query
ERROR 4097: No projections eligible to answer query
ERROR 4878: Table "string" has an out-of-date super projection "string"
ERROR 6774: Failed to create default key projections for table "string"."string": string
ERROR 7126: Table "string" (non-anchor in prejoin) has an out-of-date prejoin projection "string"
ERROR 7142: Failed to create hcatalog schema: string
ERROR 7876: Table "string" has an out-of-date projection "string"

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V16

This topic lists the errors associated with the SQLSTATE 42V16.

SQLSTATE 42V16 Description

ERRCODE_INVALID_TABLE_DEFINITION

Error messages associated with this SQLState

ERROR 2104: A table cannot have only IDENTITY/AUTO-INCREMENT columns
ERROR 2420: Cannot drop the constraint. (Table "string" has a foreign key constraint referencing the specified primary key constraint)
ERROR 2421: Cannot drop the constraint. (There is at least one prejoin projection dependent on the specified foreign key constraint)
ERROR 2622: Column "string" cannot be declared SETOF
ERROR 2626: Column "string" from table "string" in the SEGMENTED BY expression is required to be present in the projection, but is not
ERROR 2712: Constraint "string" for relation "string" already exists
ERROR 3508: IDENTITY/AUTO-INCREMENT columns are not allowed in temporary tables
ERROR 3874: Local temporary table constraint cannot reference a non-local table
ERROR 3901: MATCH types other than SIMPLE (the default) are not supported for foreign key constraints
ERROR 3907: Multiple primary keys for table "string" are not allowed
ERROR 4162: Non-local table constraint cannot reference a local temporary table
ERROR 4229: ON DELETE actions other than NO ACTION are not supported for foreign key constraints
ERROR 4234: ON UPDATE actions other than NO ACTION are not supported for foreign key constraints
ERROR 4413: Primary constraint for relation "string" already exists
ERROR 4415: Primary keys not specified
ERROR 4469: Projection anchor table is not partitioned
ERROR 4550: Referenced primary key constraint does not exist
ERROR 4876: Table "string" does not exist
ERROR 4881: Table "string" is not partitioned
ERROR 4899: Table is not partitioned
ERROR 4900: Table must have at least one column
ERROR 5269: Unsupported constraint type
ERROR 5548: Constraint not supported for user defined type column string
ERROR 5552: Correlation constraint not supported for user defined types
ERROR 5874: Failed to add table string of hcatalog schema string to catalog: no columns
ERROR 5876: Failed to alter table string of hcatalog schema string to catalog: no columns
ERROR 5879: Failed to describe hcatalog table
ERROR 5948: Local temporary objects may not specify a schema name
ERROR 6171: Cannot merge multiple partitions on an aggregate projection
ERROR 6396: SEGMENTED BY expression contains expressions not present in the SELECT list
ERROR 6434: TM task is not applicable to projections with aggregates
ERROR 6839: Key constraints on an external table cannot be enabled or disabled
ERROR 7122: Staging table's partition key and target table's partition key have different datatype
ERROR 7233: Check constraint 'string' does not reference any columns in the table
ERROR 7234: Check constraint 'string': references to column 'string' are not allowed in check predicates
ERROR 7235: Check constraint 'string': subqueries are not allowed in check predicates
ERROR 7236: Check constraint predicate is too long; value bytes, maximum length is value
ERROR 7237: Check constraints on an external table cannot be enabled or disabled
ERROR 7238: Column "string" referenced by check constraint 'string' is not in the table
ERROR 7260: Function "string" referenced by check constraint 'string' is a meta-function
ERROR 7261: Function "string" referenced by check constraint 'string' is an aggregate function
ERROR 7262: Function "string" referenced by check constraint 'string' is not immutable
ERROR 7263: Function "string" referenced by check constraint 'string' returns a set rather than a single value
ERROR 8091: Failed to parse JSON format Hive table description string: string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V17

This topic lists the errors associated with the SQLSTATE 42V17.
SQLSTATE 42V17 Description

ERRCODE_INVALID_OBJECT_DEFINITION

Error messages associated with this SQLState

ERROR 2387: Cannot create projections involving external table string
ERROR 3075: Design type string is invalid
ERROR 3078: Optimization objective string is invalid
ERROR 3199: Error during deployment querying deployment projections table for workspace string
ERROR 3200: Error during deployment querying design projections table for design string in workspace string
ERROR 3201: Error during deployment while querying deployment projections table for workspace string
ERROR 3204: Error during drop design from deployment for workspace string
ERROR 3206: Error during extend catalog while querying deployments table for workspace string
ERROR 3207: Error during getDesignTablesFromDeployment in workspace string
ERROR 3208: Error while checking whether there are only incremental design deployed for deployment string in workspace string
ERROR 3269: Error during extend catalog while querying deployments table for workspace string
ERROR 3968: More than one IDENTITY/AUTO_INCREMENT column defined for table "string"
ERROR 3983: Multiple instances of deployment string in workspace string
ERROR 4128: No valid projections found
ERROR 4230: ON DELETE rule may not use NEW
ERROR 4231: ON INSERT rule may not use OLD
ERROR 4232: ON SELECT rule may not use NEW
ERROR 4233: ON SELECT rule may not use OLD
ERROR 4635: Rule WHERE condition may not contain references to other relations
ERROR 4636: Rules with WHERE conditions may only have SELECT, INSERT, UPDATE, or DELETE actions
ERROR 4919: Temporary table projections are not allowed for this operation
ERROR 4928: There is no deployment string in workspace string
ERROR 4989: This function cannot be called on design string located in design workspace string
ERROR 4990: This function cannot be called on design string, when its design mode is string
ERROR 5367: User defined aggregate must return exactly one column. Function string returns value
ERROR 5369: User defined analytic must return exactly one column
ERROR 5384: User defined transform must provide names or aliases for return columns
ERROR 5385: User defined transform must return at least one column
ERROR 5527: An error occurred on node string when setting up the type, message:

string
ERROR 5721: Purge is not allowed on temporary tables
ERROR 6095: UseLongStrings has been deprecated
ERROR 6166: Cannot create top-k projection: projection columns and limit are too big for the top-k buffer
ERROR 6483: Invalid enum value for parameter name
ERROR 6627: Aggregate projections may only contain User Defined Transforms with partition by BATCH in the outer query
ERROR 6628: Aggregate projections may only contain User Defined Transforms with partition by PREPASS or BATCH
ERROR 6659: Can not create prejoin projection
ERROR 6660: Can not create prejoin projection when node is recovering
ERROR 6970: Subqueries in aggregate projections may only contain User Defined Transforms with partition by PREPASS
ERROR 6993: The batch and prepass User Defined Transform Functions' signatures are not compatible for use in a live aggregate projection
ERROR 6994: The batch User Defined Transform Function does not have identical input and output signature
ERROR 7788: Only projections with single phase PREPASS User Defined Transforms can have user specified sort order or segmentation
ERROR 7789: ORDER BY / SEGMENTED BY / UNSEGMENTED are not allowed in projections with two phase User Defined Transforms. The projection is automatically ordered and segmented on partition by columns

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V18

This topic lists the errors associated with the SQLSTATE 42V18.

SQLSTATE 42V18 Description

ERRCODE_INDETERMINATE_DATATYPE

Error messages associated with this SQLState

ERROR 2847: Could not determine data type of column $value
ERROR 3609: Interval must be single datetime field

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 42V21

This topic lists the errors associated with the SQLSTATE 42V21.

SQLSTATE 42V21 Description

ERRCODE_UNDEFINED_PROJECTION

Error messages associated with this SQLState

ERROR 2311: Can't find projection value
ERROR 2430: Cannot find projection column value
ERROR 3005: DDL statement interfered with refresh operation
ERROR 3736: Invalid projection name
ERROR 3737: Invalid projection name string
ERROR 4452: Projection "string" does not exist or was just dropped
ERROR 4474: Projection does not exist
ERROR 4905: Table or projection "string" does not exist
ERROR 5563: DDL statement interfered with this operation
ERROR 7528: Projection "string" does not exist
ERROR 7863: Can't find projection: value
ERROR 8223: No up-to-date super projections available to refresh projection

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V24

This topic lists the errors associated with the SQLSTATE 42V24.
SQLSTATE 42V24 Description

ERRCODE_UNDEFINED_USER

Error messages associated with this SQLState

ERROR 7213: Current user no longer exists

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V25

This topic lists the errors associated with the SQLSTATE 42V25.

SQLSTATE 42V25 Description

ERRCODE_PATTERN_MATCH_ERROR

Error messages associated with this SQLState

ERROR 2227: Argument to test_pattern_event_eval must be > 0 and less than the total number of events
ERROR 2228: Argument to test_pattern_event_eval must be a constant
ERROR 2553: Cannot use more than one pattern
ERROR 2555: Cannot use pattern test functions with pattern match functions
ERROR 3025: Defining more than 52 events is not supported
ERROR 3288: Event "string" in PATTERN clause is not defined in the DEFINE clause
ERROR 3289: Event ANY_ROW cannot be used under *, +, ?, or | when the select list contains the pattern function event_name()
ERROR 3290: Event ANY_ROW is a reserved event and cannot be user defined
ERROR 3294: Event expressions cannot contain analytic functions
ERROR 3295: Event expressions cannot contain correlated expressions
ERROR 3296: Event expressions cannot contain subqueries
ERROR 3297: Event name "string" defined more than once
ERROR 4353: Pattern events must be mutually exclusive
ERROR 4354: Pattern match query cannot contain having clause, group clause, aggregates, or distinct
ERROR 4355: Pattern match query cannot contain timeseries clause
ERROR 4356: Pattern matching recursion limit reached
ERROR 4358: PatternMatchingMaxPartition must be greater than 0
ERROR 4359: PatternMatchingMaxPartitionMatches must be greater than 0
ERROR 4494: Queries with user-defined transform functions (string) cannot have a MATCH clause
ERROR 4507: Query with analytic function string cannot have a MATCH clause
ERROR 4509: Query with pattern matching function string must include a MATCH clause
ERROR 4605: RESULTS GROUPED BY MATCH is not supported
ERROR 5283: Unsupported pattern operator
ERROR 7607: Pattern matching DFA Algorithm work space size limit reached
ERROR 8585: Pattern matching just-in-time processing stack memory limit reached

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 42V26

This topic lists the errors associated with the SQLSTATE 42V26.

SQLSTATE 42V26 Description
ERRCODE_DUPLICATE_NODE

Error messages associated with this SQLState
ERROR 4058: New node matches existing node string
ERROR 4063: New values for node string matches existing node string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 53000

This topic lists the errors associated with the SQLSTATE 53000.

SQLSTATE 53000 Description

ERRCODE_INSUFFICIENT_RESOURCES

Error messages associated with this SQLState

ERROR 2245: Attempted to create too many ROS containers for projection string
ERROR 2843: Could not create thread for recoverProjectionLocal
ERROR 2844: Could not create thread for SubsessionHandler
ERROR 2845: Could not create thread for SubsessionHandler Hurry
ERROR 2997: DBDesigner memory usage (value bytes) exceeded system limit
ERROR 3300: Exceeded temp space cap, requested value with value remaining (used value) bytes
ERROR 3416: Filter tried to allocate too much memory (value, out of value allowed)
ERROR 3587: Insufficient resources to execute plan on pool string [string]
ERROR 3921: MemoryPool string used more memory than allowed
ERROR 3937: MIN/MAX window function could not operate in memory
ERROR 4305: Out of system WOS memory during catalog SELECT
ERROR 4764: Source tried to allocate too much memory (value, out of value allowed)
ERROR 5000: Thread limit value, but statement needs value threads
ERROR 5001: ThreadManager failed to create thread string: string
ERROR 5022: Timer service failed to run value: string
ERROR 5065: Too many ROS containers exist for the following projections: string
ERROR 5921: Insufficient memory available for database designer
ERROR 5924: Insufficient resources to get resource from string pool [string]
ERROR 6941: Result set size (value KB) is too big. Try increasing TempSpaceCap (currently value KB)
ERROR 7423: Failed to acquire resources for blob 'string'
ERROR 7700: Attempted to move/copy too many ROS containers for projection string
ERROR 8178: Tried to expand to value from value, while limit is only value
ERROR 8220: Tried to expand to value from value, while limit is only value
Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 53100

This topic lists the errors associated with the SQLSTATE 53100.

SQLSTATE 53100 Description

ERRCODE_DISK_FULL

Error messages associated with this SQLState

ERROR 2475: Cannot rebalance cluster. Insufficient disk space on the following nodes: string
ERROR 2927: Could not write to [string]: string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 53200

This topic lists the errors associated with the SQLSTATE 53200.

SQLSTATE 53200 Description

ERRCODE_OUT_OF_MEMORY

Error messages associated with this SQLState

ERROR 2296: Calloc of value bytes for string failed

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
ERROR 2344: Cannot allocate sufficient memory for COPY statement (value requested, value permitted)
ERROR 3499: Hash table out of memory
ERROR 3811: Join [string] inner partition did not fit in memory; value [string]
ERROR 3813: Join did not fit in memory
ERROR 3814: Join inner did not fit in memory
ERROR 3815: Join inner did not fit in memory [string]
ERROR 3819: Join table did not fit in memory
ERROR 3895: Malloc of value bytes for string failed
ERROR 4176: Not enough memory for test directive numTopKHeaps
ERROR 4302: Out of memory
ERROR 4303: Out of memory when expanding glob: string
ERROR 4357: Pattern partition will not fit into memory
ERROR 4381: Plan memory limit exhausted: [string]
ERROR 4495: Query value exceeded memory usage limit. Design result for this query might be suboptimal
ERROR 4512: Ran out of WOS memory during string
ERROR 4524: Realloc of value bytes for string failed
ERROR 5062: Too many hash table entries
ERROR 5063: Too many matches in a single partition
ERROR 5147: Unable to reserve memory (value K) for the WOS
ERROR 5952: Malloc of value bytes in Block Memory Manager failed
ERROR 7979: Cannot allocate all variables

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

## Error Messages Associated with SQLSTATE 54000

This topic lists the errors associated with the SQLSTATE 54000.

### SQLSTATE 54000 Description

ERRCODE_PROGRAM_LIMIT_EXCEEDED

### Error messages associated with this SQLState

ERROR 2052: string Row size value is too large
Error Messages Associated with SQLSTATE 54001

This topic lists the errors associated with the SQLSTATE 54001.

SQLSTATE 54001 Description

ERRCODE_STATEMENT_TOO_COMPLEX

Error messages associated with this SQLState

ERROR 4588: Request size too big. Please try to simplify the query
ERROR 4963: The query contains a SET operation tree that is too complex to analyze
ERROR 4964: The query contains an expression that is too complex to analyze
ERROR 7913: MLA CUBE has too many columns: value
ERROR 7914: MLA grouping sets have value elements taking too much memory: value bytes
ERROR 7915: Too many grouping sets generated: value
ERROR 7916: Too many grouping sets generated: value
ERROR 8031: Too many grouping elements: value, request size too big
ERROR 8037: Query plan is too large to serialize

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 54011

This topic lists the errors associated with the SQLSTATE 54011.

SQLSTATE 54011 Description

ERRCODE_TOO_MANY_COLUMNS

Error messages associated with this SQLState

ERROR 2106: A table/projection/view can only have up to value columns -- this create statement has value
ERROR 2118: Adding column causes row size (value) to exceed MaxRowSize (value)
ERROR 2136: Aggregate function cannot have value input argument(s)
ERROR 2137: Aggregate function cannot have value return value(s)
ERROR 2181: Analytic function cannot have value return value(s)
ERROR 2291: Call to ColumnTypes.addAny() is not allowed in Aggregate functions
ERROR 3466: Function cannot have value return value(s)
ERROR 4202: Number of columns (value) exceeds limit (value)
ERROR 4481: Projection row size (value) exceeds MaxRowSize (value)
ERROR 4630: Row size exceeds MaxRowSize: value > value
ERROR 4875: Table "string" can only have up to value columns -- adding one will exceed this limit
ERROR 5898: File system cannot have value input argument(s)
ERROR 5899: File system cannot have value return value(s)
Error Messages Associated with SQLSTATE 54023

This topic lists the errors associated with the SQLSTATE 54023.

SQLSTATE 54023 Description

ERRCODE_TOO_MANY_ARGUMENTS

Error messages associated with this SQLState

ERROR 2441: Cannot have more than value segmentation columns
ERROR 2469: Cannot pass more than value arguments to a function
ERROR 4431: Procedures cannot have more than value parameters
ERROR 4646: Scalar/Transform functions cannot have more than value parameters
ERROR 5055: Too many arguments
ERROR 5056: Too many arguments to evaluate_delete_performance function

Error Messages Associated with SQLSTATE 55000

This topic lists the errors associated with the SQLSTATE 55000.

SQLSTATE 55000 Description

ERRCODE_OBJECT_NOT_IN_PREREQUISITE_STATE
Error messages associated with this SQLState

ERROR 2088: A concurrent load into the partition or a concurrent mergeout operation interfered with this statement
ERROR 2143: AHM advanced beyond snapshot epoch
ERROR 2149: AHM can't be set
ERROR 2150: AHM can't be set while retentive refresh is running
ERROR 2151: AHM can't be set. (value nodes are down, out of value.)
ERROR 2152: AHM can't be set. (value nodes are down.)
ERROR 2159: All nodes must be UP to rebalance a cluster
ERROR 2163: Already released
ERROR 2175: An error occurred when loading library file on node string, message: 
  
  string
ERROR 2200: AnalyzeStatsPlanMaxColumns configuration parameter 'value' invalid; must be greater than zero
ERROR 2201: AnalyzeStatsSampleBands configuration parameter 'value' invalid; must be greater than zero
ERROR 2241: Attempt to create view using an invalid relation
ERROR 2242: Attempt to run multi-node KV plan
ERROR 2294: CALL_USE_SESSION_NODES used without setting nodes
ERROR 2303: Can not tell if tables have data, too few responses (value) to be conclusive
ERROR 2316: Can't match imported node 'string' to node in current database
ERROR 2371: Cannot commit DML/DML while a node is shutting down
ERROR 2378: Cannot convert column "string" to type "string"
ERROR 2380: Cannot create a library without an initialized LibraryPath on node: string
ERROR 2388: Cannot create projections on a temporary table that has data
ERROR 2409: Cannot drop any more columns in string
ERROR 2410: Cannot drop column "string" since it is referenced in the default expression of column "string"
ERROR 2413: Cannot drop column "string" since it was referenced in the default expression of added column "string"
ERROR 2422: Cannot Drop: string string depends on string string
ERROR 2424: Cannot execute query with temporary table because a node has recovered since the start of this session
ERROR 2448: Cannot issue this command in a read-only transaction
ERROR 2459: Cannot modify temporary table string because a node has recovered or rebalance data took place since the start of this string
ERROR 2483: Cannot remove snapshots without an initialized SnapshotPath
ERROR 2496: Cannot revoke EXECUTE permission from the owner: string
ERROR 2497: Cannot revoke EXECUTE permission from the super user
ERROR 2505: Cannot set column "string" in table "string" to NOT NULL since it contains null values
ERROR 2512: Cannot set memoryCap for session whose current user has been dropped
ERROR 2516: Cannot set runTimeCap for session whose current user has been dropped
ERROR 2517: Cannot set tempSpaceCap for session whose current user has been dropped
ERROR 2541: Cannot use addAny() with any other input column types
ERROR 2542: Cannot use addAny() with any other output column types
ERROR 2563: Cannot validate DV storage
ERROR 2564: Cannot validate storage
ERROR 2587: Changes cannot be made to [string]. It has been retired
ERROR 2762: COPY: Cannot load into IDENTIFY column "string"
ERROR 2763: COPY: Cannot specify parsing options for IDENTIFY column "string"
ERROR 2903: Could not reset epoch because DML locks are held
ERROR 2904: Could not reset epoch because projections exist
ERROR 2933: Couldn't force partition projection string
ERROR 2934: Couldn't force partition projections string
ERROR 2954: Current phase of recovery failed due to missed event at epoch value
ERROR 2955: Current set of up nodes do not satisfy dependencies
ERROR 2956: Current set of up nodes do not satisfy dependencies for table string
ERROR 2961: Current user has been dropped so no defaults are available
ERROR 2962: Current user has been dropped so no roles are available
ERROR 2969: Cursor can only scan forward
ERROR 3002: DDL statement interfered with alter column type
ERROR 3136: drop_partition failed for string on node string. The projection contains unpartitioned data
ERROR 3196: Error deserializing objects
ERROR 3229: Error loading library file:[string]
ERROR 3254: Error reading from file
ERROR 3278: Error writing to file
ERROR 3318: Execution aborted by node state change
ERROR 3392: Failed to update local min/max objects for column "string"
ERROR 3807: JobTracker::getMarkedStorages(): Unknown job value
ERROR 3808: JobTracker::jobComplete(string): Unknown job value
ERROR 3809: JobTracker::setDetails(value,value,value): Unknown job value
ERROR 3810: JobTracker::setJobDescription(string): Unknown job value
ERROR 3838: Key value already in use
ERROR 3882: Location cannot be dropped as it stores data files
ERROR 3911: maxMemorySize for string can be changed only when the string WOS is empty
ERROR 3924: merge_partitions() failed on string because of unpartitioned data
ERROR 3925: Mergeout failed: projection string is not up-to-date
ERROR 4032: Naming conflict: string exists
ERROR 4092: No plan received at node
ERROR 4120: No transaction running on node
ERROR 4127: No valid cache found
ERROR 4138: Node string is not available for queries
ERROR 4144: Node has not been set up for plan execution
ERROR 4146: Node is not active or recovering, cannot plan query
ERROR 4148: Node not prepared to accept plan
ERROR 4151: Node unprepared for rebalance
ERROR 4177: Not enough nodes are up for Projection <string> to be available, marking it as out of date
ERROR 4483: Portal "string" cannot be run
ERROR 4457: Projection string checkpoint epoch lags snapshot epoch
ERROR 4458: Projection string contains data in the WOS
ERROR 4459: Projection string create epoch is greater than the epoch in the query
ERROR 4462: Projection string has HSE > snapshot epoch and buddy string has HSE <= snapshot epoch
ERROR 4464: Projection string is not up-to-date
ERROR 4467: Projection (name: string, oid: value) is newly added during current recovery
ERROR 4485: Projections string contain data in the WOS
ERROR 4530: Rebalance unable to moveout all data on projection string
ERROR 4592: reset_epoch is disabled because the EnableResetEpoch configuration parameter is 0
ERROR 4611: Returned string value '[string]' with length [value] is greater than declared field length of [value] of field [string] at output column index [value]
ERROR 4698: Sequence "string" has been created by an IDENTITY/AUTO_INCREMENT column and cannot be dropped
ERROR 4700: Sequence string has not been accessed in the session
ERROR 4765: Specified k-safety for projection creation is insufficient to support currently down nodes
ERROR 4791: Storage extends beyond specified segment range
ERROR 4793: Stream error: string
ERROR 4860: System is not k-safe. DDL is disallowed
ERROR 4861: System is not k-safe. DDL/DML is disallowed
ERROR 4879: Table "string" has projections in non-up-to-date state
ERROR 4880: Table "string" has projections that are not up-to-date that can refresh from buddy
ERROR 4903: Table or projection no longer exists
ERROR 4934: The attribute "string" in table "string" needs to be included in projection "string" because it is used in the partitioning expression
ERROR 4941: The data type requires length/precision specification
ERROR 4965: The restore violates K safety
ERROR 4972: The types/sizes of source column (index value, length value) and destination column (index value, length value) do not match
ERROR 5084: Tried to add field 'string' that already exists
ERROR 5085: Tried to add unknown node 'string' to user-defined query plan
ERROR 5132: Unable to evaluate the delete performance after dropping this column for projection "string"
ERROR 5151: Unable to validate data in string: string
ERROR 5204: Unknown data type
ERROR 5210: Unknown object: string
ERROR 5321: Usage of [string] cannot be changed. It has been retired
ERROR 5381: User Defined Scalar Function can only have 1 return column, but value is provided
ERROR 5522: A concurrent operation interfered with this statement
ERROR 5534: Can't create table in specified target schema
ERROR 5535: Can't find target table's schema
ERROR 5543: Cannot use column type 'any' with any other input column types
ERROR 5544: Cannot use column type 'any' with any other output column types
ERROR 5568: DVWos can not be moved
ERROR 5583: Fault Group "string" already exists in a fault group
ERROR 5587: Fault Group "string" not found in Fault Group "string"
ERROR 5590: Found value unsegmented projections with basename string; inconsistent with permanent nodes count value
ERROR 5626: Node "string" already exists in a fault group
ERROR 5627: Node "string" not found in Fault Group "string"
ERROR 5660: Source table can not be temp, virtual, system, or external
ERROR 5662: Storage tier string has not been found on all nodes
ERROR 5666: Table "string" has prejoin projections
ERROR 5667: Target table can not be temp, virtual, system, or external
ERROR 5674: TM interfered with this statement
ERROR 5705: Dvmergeout failed: projection string is not up-to-date
ERROR 5712: JobTracker::reportStart: Unknown job value
ERROR 5735: Tier string is referenced by storage policies. Can not make storage location changes as requested
ERROR 5760: Can only change setting when all started nodes are UP
ERROR 5765: Cannot change control node away from self because other nodes depend on this node to be their control node
ERROR 5766: Cannot change final control node away from self until at least one other node is promoted to be a control node
ERROR 5772: Cannot manually alter automatically generated fault groups
ERROR 5786: Column value does not have corresponding storages yet. A concurrent add column operation might be running
ERROR 5883: Failed to list hcatalog tables
ERROR 6001: Recovery failed because DVROS straddles discard epoch
ERROR 6002: Recovery failed because ROS value [0xvalue, 0xvalue] straddles endEpoch value to discard
ERROR 6003: Recovery failed because ROS straddles discard epoch
ERROR 6035: Table "string" has no non-null records under the column key_name
ERROR 6037: Table "string_string" cannot be found or was not created internally
ERROR 6065: Tried to allocate and initialize a value-byte string with value zero bytes; VString is too small
ERROR 6066: Tried to copy a value-byte string to value-byte VString object; VString is too small
ERROR 6105: View "string" is already linked to flex table "string". Linked views will not be overwritten
ERROR 6106: View "string" is already linked to this table. Linked views will not be overwritten
ERROR 6107: View "string_string" cannot be found or was not created internally
ERROR 6110: A design/deployment process is currently executing in this design space
ERROR 6121: A concurrent operation interfered with this statement
ERROR 6176: Cannot replace node string because it is already a STANDBY
ERROR 6177: Cannot replace node string because it is not DOWN
ERROR 6184: Cannot swap partition between same table
ERROR 6185: Cannot transition node string to string because it still has data
ERROR 6186: Cannot transition node string to STANDBY because its loss would cause the cluster to shutdown
ERROR 6207: Could not create internal data-storage directory 'string': value
ERROR 6357: No standby nodes are currently available
ERROR 6359: Node string has not been replaced
ERROR 6360: Node string is a string node and cannot store data
ERROR 6374: Original node string is not currently available in STANDBY mode
ERROR 6419: Table string can not be temp, virtual, system, or external
ERROR 6422: Target node string is a string node, not a STANDBY node
ERROR 6424: Target standby node string is not currently available
ERROR 6446: UDx set BOOLEAN column value to non-boolean value value
ERROR 6478: Can only take object-level snapshot from local storage locations
ERROR 6484: Mergeout failed: ROS(es) have been dropped/moved
ERROR 6524: DDL interfered with this statement. Table is not partitioned or partition expression got changed
ERROR 6553: Compact storage failed: projection string is not up-to-date
ERROR 6554: Compact storage failed: ROS(es) have been dropped/moved
ERROR 6633: An enabled constraint can only be declared on a global temporary table during CREATE TABLE
ERROR 6634: An enabled constraint cannot be created on a temporary table with existing data
ERROR 6646: Attempted to commit when column value has value rows while column 0 has value rows
ERROR 6647: Attempted to use a string as a string
ERROR 6648: Attempted to write past the end of a column
ERROR 6658: Can not string to the same table
ERROR 6665: Can not set priority for text index, please use its source table
ERROR 6667: Can only change setting when lockless recovery is enabled
ERROR 6675: Cannot drop column "string" since it is the last non-IDENTITY, non-AUTO_INCREMENT column
ERROR 6677: Cannot execute query because table recovery status change
ERROR 6784: Final attempt at a database snapshot upgraded storage ids on the following nodes: string
ERROR 6811: Incorrect use of setter in processBlock
ERROR 6812: Incorrect use of setter in processPartition for [value] column
ERROR 6826: Invalid node Oid value
ERROR 6870: Multiple values for the parameter string. The parameter will not be set
ERROR 6937: Restore: Cannot overwrite object string
ERROR 6957: Source items are not the correct encoding for direct copies
ERROR 6958: Source items are not the correct size for direct copies
ERROR 6976: Table "string" has not been recovered. Please try later
ERROR 6982: Table can not be temp, virtual, system, or external
ERROR 6989: Terminate() must be overridden for a User Defined Aggregate
ERROR 6997: The key string doesn't exist
ERROR 7001: The sessionParamReader for namespace 'string' doesn't exist
ERROR 7002: The sessionParamWriter for namespace 'string' doesn't exist
ERROR 7016: UDx set BOOLEAN column value, row value to non-boolean value value
ERROR 7017: Unable to acquire side process info
ERROR 7072: WebHCat query [string] failed: string
ERROR 7093: Comment length of parameter 'string' is 'value' which exceeds the maximum allowed 'value'
ERROR 7096: Failed to rollover MinMaxObj on all nodes
ERROR 7119: Source and target table do not match: string
ERROR 7121: Staging table and target table do not match: string
ERROR 7173: Current set of up nodes do not satisfy dependencies for DFS file distribution value
ERROR 7193: RecoverByContainer::recover can’t advance the cpe by recovering containers
ERROR 7212: Cannot restore data to node string
ERROR 7290: Trying to set the column "string" to size of value All data type lengths in table "string" must be greater than value, the current maximum raw size for flex table values
ERROR 7299: Cannot alter type of column "string" since it is referenced in the default expression of table "string", column "string"
ERROR 7300: Cannot alter type of column "string" since it is referenced in the set using expression of table "string", column "string"
ERROR 7301: Cannot drop column "string" since it is referenced in the default expression of table "string", column "string"
ERROR 7302: Cannot drop column "string" since it is referenced in the set using expression of table "string", column "string"
ERROR 7343: string expression of IDENTITY/AUTO_INCREMENT column "string" cannot be altered
ERROR 7390: Cannot set idlesessiontimeout for session whose current user has been dropped
ERROR 7448: Location cannot be dropped as it stores DFS files
ERROR 7458: Mergeout failed: found missed string on table string
ERROR 7537: Sequence "string" has been created by an IDENTITY/AUTO_INCREMENT column and cannot be used in a string expression
ERROR 7583: User or Role value cannot be found
ERROR 7619: SBJobTracker::setAdditionalInfo: Unknown oid "value"
ERROR 7671: Wrong checksum for library file string
ERROR 7766: Cannot drop column "string" since it is referenced in the set using expression of column "string"
ERROR 7811: SendFiles on node string: file [string] has changed
ERROR 7821: Cannot rename user "string" since they use MDS password format
ERROR 7856: Not all requested nodes are available at this moment
ERROR 7857: Not enough available nodes at this moment
ERROR 7866: DDL statement interfered with this statement. ProjCol value cannot be found
ERROR 7978: Can only use KV hint with SELECT queries
ERROR 7983: Cannot use KV Hint with a subquery
ERROR 8009: Running KV query inside existing transaction. There will be a performance implication. Consider rerunning the query outside of an existing transaction
ERROR 8061: Cannot create COMMUNAL storage location with usage string
ERROR 8062: Cannot create COMMUNAL storage location without sharding enabled
ERROR 8064: Cannot drop last subscription for a shard
ERROR 8069: Cannot set grace period for session whose current user has been dropped
ERROR 8083: Database does not have shards to rebalance
ERROR 8139: Projection must be created in the same schema as its anchor table
ERROR 8147: SALColumn found during restore. Restoring objects from on older vertica version is not supported
ERROR 8151: Session resource pool does not exist
ERROR 8159: Size can be changed only for DEPOT locations
ERROR 8173: Text index must be created in the same schema as its source table
ERROR 8181: Unable to read file string
ERROR 8185: Usage of [string] cannot be changed. It is a depot location
ERROR 8186: User lacks privileges on resource pool 'string'
ERROR 8210: Object value appears in the newborns of the Transaction Catalog.

Object is Global: string
Object has Shard Tag: value
Object was created in this TXN tier: string
ERROR 8222: No active subscriptions found for the session
ERROR 8228: Cannot load snapshot containing DOWN source node string into the current database
ERROR 8229: Cannot restore or replicate while a node is recovering
ERROR 8232: No snapshot to load in the current session. Did load_snapshot_prep() succeed?
ERROR 8235: Sequence "string" has been created by an IDENTITY/AUTO_INCREMENT column and name is internal to Vertica, can not be changed directly
ERROR 8236: Sequence "string" has been created by an IDENTITY/AUTO_INCREMENT column, the schema associate with it cannot be changed directly
ERROR 8237: Snapshot contains internally inconsistent epochs; unable to load
ERROR 8239: Unable to map excluded source node string
ERROR 8257: Cannot find an UP node with an ACTIVE subscription to the replica shard
ERROR 8261: Dvmergeout failed: not a participating subscriber
ERROR 8266: Mergeout failed: not a participating subscriber
ERROR 8305: Partition projection failed: not a participating subscriber
ERROR 8313: ROS/shard inconsistencies during partitioning
ERROR 8347: No response from required nodes: string. Resubmit operation later
ERROR 8348: Shards string are missing participating subscription
ERROR 8360: Need to split partitions before copy/move/swap partitions
ERROR 8365: Unable to copy/move/swap partitions because some projection(s) contain unpartitioned data
ERROR 8384: ROS/shard inconsistencies during dvmergeout
ERROR 8385: ROS/shard inconsistencies during mergeout
ERROR 8400: Can not partition projection by given range due to the group by expression
ERROR 8416: Node string was not found on communal storage
ERROR 8423: Recovery failed due to missed event at epoch value
ERROR 8430: Some storage containers must be split before the specified partition range can be dropped
ERROR 8435: The number of nodes in communal storage (value) is less than the number of nodes specified (value)
ERROR 8445: Unable to moveout all data on projection string
ERROR 8453: Rebalance: Refreshing projection string
ERROR 8459: Node string is shut down
ERROR 8470: Attempting to load snapshot with inconsistently applied upgrade tasks
ERROR 8481: Database cannot be revived from a configuration file created by a different version. The current database version is string, while the configuration file version is string

ERROR 8482: DVWOS found. Try again

ERROR 8483: Execution aborted by cluster configuration changes at catalog version value, more recent than our latest sync at version value on this node

ERROR 8484: File string does not exist

ERROR 8495: WOS found. Try again

ERROR 8501: Cannot create miniroses on node string

ERROR 8528: Subscription identified by (string,string) is already marked for removal

ERROR 8562: Node failure during planning

ERROR 8567: ROSs were moved/deleted during TM operation

ERROR 8570: Can’t find the opened file [string]

ERROR 8592: Storage container value of table string has been moved to table string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 55006

This topic lists the errors associated with the SQLSTATE 55006.

SQLSTATE 55006 Description

ERRCODE_OBJECT_IN_USE

Error messages associated with this SQLState

ERROR 2060: string WOS is not empty; cannot renew. Do a moveout
ERROR 2307: Can’t drop self
ERROR 3003: DDL statement interfered with Database Designer
ERROR 3004: DDL statement interfered with query replan
ERROR 3896: Manual mergeout not supported while tuple mover is running
ERROR 3897: Manual moveout not supported while tuple mover is running
ERROR 4122: No up-to-date super projection left on the anchor table of projection string
ERROR 4139: Node string transitioned to state UP during this statement
ERROR 4145: Node is active and cannot be altered
ERROR 4455: Projection string cannot be dropped because K-safety would be violated
ERROR 4470: Projection cannot be dropped because history after AHM would be lost
ERROR 4488: Projections cannot be dropped or data would be lost due to down nodes
ERROR 4527: Rebalance is already running
ERROR 4528: Rebalance is already scheduled to run in the background
ERROR 4882: Table "string" is used as a dimension in a prejoined projection
ERROR 4896: Table (value) has been dropped
ERROR 4971: The status of one or more nodes changed during query planning
ERROR 6052: The system must retain at least one control node after the drop
ERROR 6162: Cannot alter a control node to be a STANDBY node
ERROR 6163: Cannot alter initiator node to STANDBY
ERROR 7219: A DDL statement interfered with this statement: constraint 'string' has been disabled or dropped on table 'string'
ERROR 7220: A DDL statement interfered with this statement: constraint 'string' has been enabled on table 'string'
ERROR 7402: DDL statement interfered with this statement. Text indices don't line up
ERROR 7616: SBJobTracker::createMarker: existing projOid value
ERROR 7888: DDL statement (string) interfered with this statement

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 55V02

This topic lists the errors associated with the SQLSTATE 55V02.

SQLSTATE 55V02 Description

ERRCODE_CANT_CHANGE_RUNTIME_PARAM

Error messages associated with this SQLState

ERROR 4324: Parameter string will not take effect until database restart
ERROR 7567: This Parameter has been disabled and will not be set

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE 55V03

This topic lists the errors associated with the SQLSTATE 55V03.

SQLSTATE 55V03 Description

ERRCODE_LOCK_NOTAVAILABLE

Error messages associated with this SQLState

ERROR 5156: Unavailable: string - Locking failure: string
ERROR 5157: Unavailable: [Txn value] string - string error string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 55V04

This topic lists the errors associated with the SQLSTATE 55V04.

SQLSTATE 55V04 Description

ERRCODE_TM_MARKER_NOTAVAILABLE

Error messages associated with this SQLState

ERROR 2082: A string operation is already in progress on projection string.string [container value txnid value session string]
ERROR 2083: A string operation is already in progress on projection string.string [txnid value session string]
Error Messages Associated with SQLSTATE 57014

This topic lists the errors associated with the SQLSTATE 57014.

SQLSTATE 57014 Description

ERRCODE_QUERY_CANCELED

Error messages associated with this SQLState

ERROR 2246: Audit canceled
ERROR 2279: Bulk Import canceled
ERROR 2310: Can’t find projection
ERROR 2325: Canceled (in string)
ERROR 2326: Canceled: string - Locking canceled: string
ERROR 2327: Canceled: [Txn value] string - string string
ERROR 2576: Catchup recovery interrupted
ERROR 2704: Connection canceled
ERROR 2996: DBDesigner canceled by user
ERROR 3246: evaluate_delete_performance canceled
ERROR 3319: Execution canceled (compile)
ERROR 3320: Execution canceled (prepare)
ERROR 3321: Execution canceled (start)
ERROR 3322: Execution canceled by operator
ERROR 3323: Execution got unlucky!
ERROR 3324: Execution intentionally failed
ERROR 3326: Execution time exceeded run time cap of string
ERROR 3515: import_catalog_objects canceled
ERROR 4114: No super projection available for analyze_statistics
ERROR 4143: Node failure in string
ERROR 4287: Operator intervention on string
ERROR 4380: Plan canceled prior to execute call
ERROR 4439: Processing aborted by peer on string
ERROR 4496: Query canceled while waiting for resources
ERROR 4787: Statement abandoned due to subsequent DDL
ERROR 4789: Statement is canceled
ERROR 4843: Subsession interrupted
ERROR 5757: build_flextable_view canceled
ERROR 5787: compute_flextable_keys canceled
ERROR 5915: Hcatalog webservices query canceled
ERROR 5953: materialize_flextable_columns canceled
ERROR 6292: Internal query raised exception during ALTER TABLE ADD CONSTRAINT
ERROR 6389: Rebalance task string canceled
ERROR 6535: Table Owner lock canceled
ERROR 7088: Can not perform the upgrade because 'EnableMinMaxScaling' configuration parameter is turned off. Please turn it on first
ERROR 7115: Receive on string: query has been canceled
ERROR 7278: StopExecution BeforePlan
ERROR 7279: StopExecution CompilePlan
ERROR 7280: StopExecution ExecutePlan
ERROR 7281: StopExecution InitPlan
ERROR 7282: StopExecution Plan
ERROR 7283: StopExecution PreparePlan
ERROR 7284: StopExecution SerializePlan
ERROR 7457: Mergeout canceled
ERROR 7462: Moveout canceled
ERROR 8467: Uploader was interrupted due to user cancel or query failure

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 57015

This topic lists the errors associated with the SQLSTATE 57015.

SQLSTATE 57015 Description

ERRCODE_SLOW_DELETE
Error messages associated with this SQLState

ERROR 5822: Detected slow delete for projection string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 57V01

This topic lists the errors associated with the SQLSTATE 57V01.

SQLSTATE 57V01 Description

ERRCODE_ADMIN_SHUTDOWN

Error messages associated with this SQLState

ERROR 3556: Initiating node is down
ERROR 4150: Node status is not UP
ERROR 8258: Initiating node is down

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 57V03

This topic lists the errors associated with the SQLSTATE 57V03.
SQLSTATE 57V03 Description

ERRCODE_CANNOT_CONNECT_NOW

Error messages associated with this SQLState

ERROR 2863: Could not fork UDx zygote process, string
ERROR 2929: Couldn't create new UDx side process, failed to get UDx side process info from zygote: string
ERROR 2930: Couldn't create new UDx side process, the language string is not supported
ERROR 2937: Couldn't set TCP_NODELAY option, might get latency in RPC message delivery: string
ERROR 3363: Failed to connect to side process, string
ERROR 3364: Failed to connect to UDx zygote, string
ERROR 3366: Failed to create new UDx side process, couldn't connect to it: string
ERROR 4720: Session manager cannot add an external session - disabled
ERROR 5699: Cannot find java binary: neither the Linux environment variable JAVA_HOME nor Vertica config parameter JavaBinaryForUDx is set
ERROR 5702: Couldn't create new UDx side process: string
ERROR 5803: Couldn't create new UDx side process, failed to set locale information: string
ERROR 6707: Could not fork UDx zygote process: string
ERROR 6712: Couldn't cancel the external procedure: string
ERROR 6715: Couldn't execute the external procedure: string
ERROR 6840: Key generation for the zygote failed: [string]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 57V04

This topic lists the errors associated with the SQLSTATE 57V04.

SQLSTATE 57V04 Description

ERRCODE_DML_COMMIT_DURING_SHUTDOWN
Error messages associated with this SQLState

ERROR 6523: Cannot commit DML while a node is shutting down

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 58030
This topic lists the errors associated with the SQLSTATE 58030.

SQLSTATE 58030 Description
ERRCODE_IO_ERROR

Error messages associated with this SQLState

ERROR 2024: string Error occurred during BZIP decompression. BZIP error code: value
ERROR 2026: string Error occurred during ZLIB decompression. ZLIB error code: value, Message: string
ERROR 2253: Bad return from WaitForMultipleObjects: value (value)
ERROR 2432: Cannot get LibraryPath from node: string
ERROR 2433: Cannot get MD5 checksum from node: string
ERROR 2600: Checksums do not match (computed=0xvalue, fromdisk=0xvalue) discarding checkpoint!
ERROR 2674: ColumnAccessBase open error
ERROR 3197: Error deserializing snapshot info from file string
ERROR 3255: Error reading from file string
ERROR 3303: Exception during measurement deserialization
ERROR 3305: Exception during Stats deserialization:string
ERROR 3370: Failed to create socket waiting event: value
ERROR 3385: Failed to reset socket waiting event: value
ERROR 3408: File size on disk does not match catalog for string
ERROR 3412: FileColumnReader: Get block string @ value error
ERROR 3478: getnameinfo_all() failed: string
ERROR 3550: Info file string does not exist
ERROR 3796: IO_ERROR writing data file [string]
ERROR 4364: Performance measurement of [string] failed
ERROR 4377: Pixw finish error
ERROR 4378: Pixw open error
ERROR 4379: Pixw write error
ERROR 4518: Read error when expanding glob: string
ERROR 4632: RowAccessBase open error
ERROR 5124: Unable to close catalog file [string]
ERROR 5126: Unable to create catalog file [string]
ERROR 5131: Unable to drop catalog file [string]
ERROR 5133: Unable to fsync catalog file [string] errno=value
ERROR 5141: Unable to open file [string]
ERROR 5152: Unable to write catalog file [string]
ERROR 5153: Unable to write checksum to catalog file [string]
ERROR 5154: Unable to write object to catalog file [string]
ERROR 5887: Failed to mount file system value: string
ERROR 5901: Filesystem does not pass basic test: string
ERROR 5902: Filesystem does not pass basic test: I/O data differ
ERROR 6074: Unable to close catalog file after fsync [string] errno=value
ERROR 6077: Unable to fsync catalog dir [string] errno=value
ERROR 6079: Unable to open catalog dir fd for fsync [string] errno=value
ERROR 6080: Unable to open catalog dir for fsync [string] errno=value
ERROR 6081: Unable to open catalog file for fsync [string] errno=value
ERROR 6082: Unable to open spread conf file string for writing
ERROR 6084: Unable to stat file string: string
ERROR 6118: string Error occurred during LZO decompression (compressed data violation). LZO error code: value
ERROR 6258: Exception during file writer deserialization: string
ERROR 6259: Exception during file writer serialization: string
ERROR 6260: Exception during snapshot deserialization: string
ERROR 6261: Exception during snapshot serialization: string
ERROR 6262: Exception during storage container deserialization: string
ERROR 6263: Exception during storage container serialization: string
ERROR 6527: Filesystem does not support snapshot
ERROR 6528: Filesystem failed to restore snapshot
ERROR 6550: Cannot open catalog source file string
ERROR 6746: Duplicate storage location id: value
ERROR 6762: Error manifest format
ERROR 6776: Failed to glob [string] because of error: string
ERROR 6808: Improperly ordered or duplicate storage ids: string, string
ERROR 6830: Invalid section for storage locations
ERROR 6860: Malformed object line: string
ERROR 6861: Malformed storage location line: string
ERROR 6874: No Library tar file: string
ERROR 6984: tar_append_file: string: value
ERROR 6985: tar_append_tree failed: value Real dir: string; save dir: string
ERROR 6986: tar_close failed: value
ERROR 6987: tar_extract_all failed: value Extract path: string
ERROR 6988: tar_open failed: value Path: string
ERROR 7178: Error loading from all sources
ERROR 7718: Exception on closing file: string
ERROR 7720: Exception on flushing file: string
ERROR 7721: Exception on opening file: string
ERROR 7722: Exception on reading file: string
ERROR 7723: Exception on resizing file: string
ERROR 7724: Exception on writing to file: string
ERROR 7728: No place to store chunk files
ERROR 7782: Cannot create Blobs directory
ERROR 7869: No files match when expanding glob: [string]
ERROR 8025: Exception when open blob file: string
ERROR 8067: Cannot open file string for tiered catalog printer
ERROR 8085: Empty filename specified
ERROR 8260: Cannot write file: import_log.json
ERROR 8271: Please specify schema name when using *
ERROR 8340: Unable to open transaction log file [string]
ERROR 8350: Error syncing missing transaction logs to [string] following node startup: string
ERROR 8351: Error syncing transaction logs to [string]: string
ERROR 8352: Failed to copy checkpoint to [string] for commitid=value at global catalog version value: string
ERROR 8353: Failed to remove value old checkpoints from string: string
ERROR 8354: Failed to remove old txn logs older than value from string: string
ERROR 8355: Failed to write cluster configuration file for catalog version value: string
ERROR 8368: Cluster configuration file [string] is empty
ERROR 8370: Error loading remote catalog: string
ERROR 8374: Unable to find cluster configuration file [string]
ERROR 8380: Txn log [string] synced from [string] is larger: (value remotely vs value locally)
ERROR 8381: Txn log [string] was only partially synced to [string]: (value remotely vs value locally)
ERROR 8396: [string] does not exist or is not a directory
ERROR 8458: Cannot write file: export_log.json
ERROR 8471: Error writing catalog diffs from string: string
ERROR 8472: Error writing catalog diffs to string: string
ERROR 8493: UploadFileTask::File string not found for uploading
ERROR 8503: Cannot get a reply from node: string
ERROR 8539: Exception during DTRosFileInfo serialization: string
ERROR 8540: Exception during RosMessage serialization: string
ERROR 8541: Exception during sending DTContainers (temp name = string)
ERROR 8542: Exception during sending EOF : string
ERROR 8543: FileColumnWriter::Exception during file name sending : string
ERROR 8551: StorageBundleWriter: Exception during file name sending : string
ERROR 8552: StorageBundleWriter: Exception during flush bundled data: string
ERROR 8553: StorageBundleWriter: Exception during footer sending : string
ERROR 8578: Exception during file name deserialization: string
ERROR 8579: Exception during ROS message deserialization: string

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE 58V01

This topic lists the errors associated with the SQLSTATE 58V01.

SQLSTATE 58V01 Description

ERRCODE_UNDEFI NED_FILE

Error messages associated with this SQLState

ERROR 3664: Invalid filename. Input filename is an empty string
ERROR 6222: Depends can specify files only. [string] is not a valid file

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE V1001

This topic lists the errors associated with the SQLSTATE V1001.
SQLSTATE V1001 Description

ERRCODE_LOST_CONNECTIVITY

Error messages associated with this SQLState

ERROR 2709: Connection to spread closed
ERROR 4048: NetworkReceive: Decompression failed
ERROR 4054: NetworkSend on string: failed to open connection to node string (string)
ERROR 4140: Node string was not successfully added to the cluster
ERROR 4142: Node failure during execution
ERROR 4533: Receive: Decompression failed
ERROR 4534: Receive on string: Message receipt from string failed [string]
ERROR 4541: ReceiveFiles on string: Unexpected end of stream from string [string]
ERROR 4547: RecvFiles on string: Open failed on node [string] (string)
ERROR 4572: RemoteSend: Open failed on node [string], IPAddr is [string], port is [value] (string)
ERROR 4683: Send: Connection not open [string tag:value plan value]
ERROR 4684: Send: Open failed on node [string] (string)
ERROR 4689: SendFiles on string: Open failed on node [string] (string)
ERROR 5579: Failure in send on socket string: string
ERROR 5624: NetworkReceive on string: failed to open connection to node string (string)
ERROR 5625: NetworkReceive on string: Message receipt from string failed: string
ERROR 5658: Send on string: Open failed on node [string] (Address lookup for string(string) failed)
ERROR 7116: Receive on string: open failed for node string (string) --- has query been cancelled?
ERROR 8537: DataTargetProxy on string: handle is canceled
ERROR 8538: DataTargetProxy on string: Unexpected end of stream from string [string]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE V1002

This topic lists the errors associated with the SQLSTATE V1002.
SQLSTATE V1002 Description

ERRCODE_K_SAFETY_VIOLATION

Error messages associated with this SQLState

ERROR 2406: Cannot drop value nodes from a value node cluster with value nodes down - cluster would appear partitioned and database would shutdown. Bring some nodes up and try again
ERROR 2529: Cannot support K=value on only value nodes
ERROR 2957: Current system KSAFE level is not fault tolerant
ERROR 4477: Projection KSAFE value can not be met with only value nodes
ERROR 4478: Projection KSAFE override value cannot be less than current system K-safe value value
ERROR 8556: Cannot drop node as with active subscribers

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE V1003

This topic lists the errors associated with the SQLSTATE V1003.

SQLSTATE V1003 Description

ERRCODE_CLUSTER_CHANGE

Error messages associated with this SQLState

ERROR 2094: A node has come UP which missed ALTER COLUMN check
ERROR 2095: A node has come UP which missed drop partition keys check
ERROR 2096: A node has come UP which missed partitioning check
ERROR 2097: A node has entered the cluster since the session started
ERROR 2098: A node has entered the cluster since the session was started
ERROR 2099: A node has entered/left the database cluster
ERROR 5312: Up node set changed during restore
ERROR 5523: A node has come UP which missed ADD COLUMN statement
ERROR 6876: No nodes up!
ERROR 7214: Node types changed during restore
ERROR 8480: A node has come UP which missed ADD COLUMN O LOCK

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE V2000

This topic lists the errors associated with the SQLSTATE V2000.

SQLSTATE V2000 Description

ERRCODE_AUTH_FAILED

Error messages associated with this SQLState

ERROR 3493: GSS error: string. Error details: (string/string)
ERROR 3718: Invalid old password
ERROR 6635: An error occurred during GSS authentication
ERROR 6636: An error occurred during kerberos authentication

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE V2001

This topic lists the errors associated with the SQLSTATE V2001.
SQLSTATE V2001 Description

ERRCODE_LICENSE_ISSUE

Error messages associated with this SQLState

ERROR 2382: Cannot create another node. The current license permits value node(s) and the database catalog already contains value node(s)
ERROR 3248: Error parsing license end date
ERROR 3863: License issue: string
ERROR 6164: Cannot alter STANDBY node. The current license permits value node(s) and the database catalog already contains value node(s)
ERROR 6549: Cannot install new license to the database. New license permits value node(s) but the database catalog already contains value node(s)
ERROR 7018: Unable to audit license: database lacks an active license!
ERROR 8263: Invalid combination of input licenses
ERROR 8268: No license found
ERROR 8277: Unable to overwrite existing licenses
ERROR 8450: Cannot drop the license. The remaining licenses permit value node(s) and the database catalog contains value node(s)
ERROR 8454: Some/All of your Vertica licenses have expired

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE VC001

This topic lists the errors associated with the SQLSTATE VC001.

SQLSTATE VC001 Description

ERRCODE_CONFIG_FILE_ERROR
Error messages associated with this SQLState

ERROR 3833: Kerberos keytab file must be owned by the database user, and have no permissions for "group" or "other"
ERROR 4951: The Kerberos keytab file is either empty or too small in size to be valid

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE VD001

This topic lists the errors associated with the SQLSTATE VD001.

SQLSTATE VD001 Description

ERRCODE_DESIGNER_FUNCTION_ERROR

Error messages associated with this SQLState

ERROR 2010: string cannot be NULL
ERROR 2012: string clause does not exist in the query
ERROR 2202: Anchor table for projection string does not exist, so it cannot be added to deployment
ERROR 2204: Anchor table of projection string is a Session scoped table
ERROR 2205: Anchor table of projection string is a System table
ERROR 2211: API string not available in old DBD engine
ERROR 2212: API cannot take query input file and query string, only one can be set
ERROR 2304: Can only load value string under the string design type
ERROR 2328: Cannot string as design was created already
ERROR 2336: Cannot add another Comprehensive design to deployment string
ERROR 2337: Cannot add design projections in extend catalog type deployment string in workspace string
ERROR 2338: Cannot add design tables to design string because there are populated designs
ERROR 2339: Cannot add design to deployment string because design string has not been populated
ERROR 2369: Cannot clear design tables from design string because there are populated designs
ERROR 2375: Cannot compute projections to be dropped for only incremental designs deployment
ERROR 2394: Cannot design encoding for Projection string as it does not have any AUTO encoded columns
ERROR 2395: Cannot design encoding for Projection string as it is not SAFE -- Create its buddies
ERROR 2396: Cannot design/deploy for virtual system schema string
ERROR 2423: Cannot execute deployment when there are non-up-to-date safe projections for table string
ERROR 2454: Cannot load invalid query: string
ERROR 2456: Cannot load queries as design was populated already
ERROR 2463: Cannot output design projections because design is not available
ERROR 2464: Cannot output query because query id is invalid
ERROR 2471: Cannot populate drop projections in extend catalog type deployment string in workspace string
ERROR 2477: Cannot refresh projections for table value as it was dropped
ERROR 2480: Cannot remove any design table from design string because there are populated designs
ERROR 2485: Cannot remove workspace string because it does not exist
ERROR 2492: Cannot retrieve design tables for design string in workspace string
ERROR 2493: Cannot retrieve information for design string in workspace string
ERROR 2507: Cannot set k-safety when design string has been populated
ERROR 2657: Column 'string' does not exist in Table string
ERROR 2658: Column 'string' is duplicated in the column list
ERROR 3053: Deployment string already exists in workspace string
ERROR 3054: Deployment got canceled
ERROR 3056: Deployment ksafty should be equal or greater than design ksafty. Deployment ksafty is value and design ksafty is value
ERROR 3057: Deployment name cannot be NULL
ERROR 3058: Deployment Projections status is set to Error
ERROR 3060: Design string already exists
ERROR 3061: Design string already exists for workspace string
ERROR 3063: Design string has already been added to deployment string
ERROR 3064: Design string has not been populated in workspace string so projection cannot be added
ERROR 3065: Design string hasn't been populated
ERROR 3066: Design string in workspace string is not available
ERROR 3067: Design string is already populated
ERROR 3068: Design string is populated, remove design first (designer_remove_design)
ERROR 3071: Design name cannot have more than value characters
ERROR 3072: Design name may contain only alphanumeric or underscore characters
ERROR 3073: Design did not complete successfully, so deployment did not start
ERROR 3074: Design K-safety should be 0
ERROR 3077: Design name cannot have character '.'
ERROR 3079: Optimization objective cannot be NULL
ERROR 3080: Design query with design_query_id value does not exist
ERROR 3081: Design Query with design_query_id string does not exist
ERROR 3082: Design string does not exist
ERROR 3087: design_override_type string for query (design_query_id value) already exists
ERROR 3088: design_override_type string for table string already exists
ERROR 3089: design_override_type string for table string does not exist
ERROR 3100: Did not find any projections to design encodings for
ERROR 3101: Did not find design projections for projection ids given
ERROR 3102: Did not find design projections for tablePattern string
ERROR 3103: Did not find design tables to add
ERROR 3104: Did not find design tables to remove
ERROR 3105: Did not find projection id value in deployment string in workspace string
ERROR 3106: Did not find projections for design string in workspace string
ERROR 3107: Did not find rows in deployment table for deployment string in workspace string
ERROR 3108: Did not find rows in designs table for workspace string
ERROR 3140: Dropping design without getting design projections, API call is of no use
ERROR 3166: Empty design name is not allowed
ERROR 3188: Error after projection refresh: string
ERROR 3194: Error creating workspace: Invalid workspace name
ERROR 3195: Error deleting deployment status table
ERROR 3202: Error during deployment while setting ksafe before deployment starts
ERROR 3203: Error during design: string
ERROR 3205: Error during drop projections: string
ERROR 3208: Error during projection creation: string
ERROR 3214: Error during remove design string
ERROR 3215: Error during rename projections: string
ERROR 3241: Error opening query input file [string]
ERROR 3250: Error querying deployment projections statements table
ERROR 3251: Error querying deployment projections table
ERROR 3252: Error querying design projections table for design string in workspace string
ERROR 3253: Error querying: string
ERROR 3266: Error status for projections to add for table string
ERROR 3267: Error status for projections to drop for table string
ERROR 3268: Error updating deployment projections table
ERROR 3270: Error while loading statistics into design tables for design string
ERROR 3277: Error writing to [string]
ERROR 3356: External table string is not a design table
ERROR 3358: Failed during select mark_design_ksafe(value)
ERROR 3415: Filename cannot be NULL
ERROR 3480: Given design string does not exist
ERROR 3489: Group-by override value on query value cannot be satisfied
ERROR 3543: Incremental override design needs a query or an input query file to be set
ERROR 3574: INSERT query without SELECT is not supported: string
ERROR 3649: Invalid Deploy Operation string string
ERROR 3650: Invalid deploy status string string
ERROR 3740: Invalid query input file [string]
ERROR 3795: Invalid design creator name
ERROR 3817: Join override value on query value cannot be satisfied
ERROR 3824: K cannot be value (maximum allowed is value)
ERROR 3825: K must be equal to or greater than value, cannot reduce current k-safety level
ERROR 3827: K-safety cannot be NULL
ERROR 3867: List of projections cannot be NULL
ERROR 3898: mark_design_ksafe(value) failed; some projections may not be k-safe
ERROR 4031: Namespace for LOCAL temporary tables cannot be used to add design tables
ERROR 4057: New kSafety cannot be less than 0
ERROR 4078: No deployment data in string.string
ERROR 4080: No drop entries found for deployment string in workspace string
ERROR 4099: No projections found for the projection ids string string
ERROR 4117: No tables found in schema string
ERROR 4118: No tables found in the table pattern given
ERROR 4119: No tables to design projections for
ERROR 4235: One of the design tables no longer exist
ERROR 4311: Override (design_override_id value) is ignored because the table string is no longer a design table
ERROR 4312: Override (design_override_id value) is ignored because the table does not exist
ERROR 4313: override_type string for query (design_query_id value) does not exist
ERROR 4314: override_type string is invalid
ERROR 4460: Projection string does not exist
ERROR 4461: Projection string does not exist
ERROR 4466: Projection string to be refreshed was dropped
ERROR 4475: Projection id cannot be NULL
ERROR 4476: Projection id list cannot be NULL
ERROR 4479: Projection name cannot be NULL
ERROR 4497: Query Id cannot be NULL
ERROR 4498: Query referencing EPOCH column is not supported
ERROR 4499: Query referencing local temporary table string is not supported: string
ERROR 4500: Query referencing projection string is not supported: string
ERROR 4501: Query without referencing any catalog table is not supported: string
ERROR 4503: Query table string does not exist
ERROR 4504: Query table contains multiple entries with qid = value
ERROR 4505: Query weight must be a positive number
ERROR 4525: Rebalance data cannot proceed when there are non-up-to-date projections in the catalog
ERROR 4526: Rebalance data failed during select mark_design_ksafe(value)
ERROR 4651: Schema string does not exist
ERROR 4652: Schema string is not a designer created schema, so it cannot be dropped
ERROR 4655: Schema name cannot be NULL
ERROR 4721: Session scoped table string is not a design table
ERROR 4783: Start deploy: deploy is already running on this node
ERROR 4819: Subqueries in UPDATE/DELETE is not supported: string
ERROR 4866: System table string is not a design table
ERROR 4874: Systems tables within system schema string cannot be added as design tables
ERROR 4885: Table string does not exist
ERROR 4886: Table string does not exist anymore in the catalog
ERROR 4888: Table string has no statistics or data. As a result, the proposed projections on this table may be suboptimal
ERROR 4890: Table string is not a design table
ERROR 4891: Table string is not a design table, referenced in query (qid=value): string
ERROR 4902: Table name cannot be NULL
ERROR 4907: Table pattern cannot be NULL
ERROR 4920: Terminated after SO enum. See log for the content of the SOs
ERROR 4942: The design table entry with table name string.string is corrupted, as that table has been renamed in the Vertica catalog
ERROR 4976: There are value nodes. Deployment K = value is not possible
ERROR 4977: There are no projections to add in deployment string for workspace string so no projections can be dropped
ERROR 4980: There is 1 node. Deployment K = value is not possible
ERROR 4981: There is more than one design string in workspace string
ERROR 4983: There is no design tables system table in workspace string
ERROR 4991: This invalid query cannot be loaded: string
ERROR 4994: This non-SELECT query is not supported: string
ERROR 4995: This query is not supported in DBDesigner
ERROR 5363: User string does not have privileges to access design table: string
ERROR 5364: User string does not have privileges to access table: string
ERROR 5390: User has insufficient privileges on table string
ERROR 5480: Workspace string cannot be a virtual system schema
ERROR 5481: Workspace string does not exist
ERROR 5482: Design string is configured for extend_catalog so no designs can be computed
ERROR 5483: Design string is configured for extend_catalog so remove drops is not supported
ERROR 5484: Design string is configured for extend_catalog so there are no design tables
ERROR 5485: Design string is configured for extend_catalog, there are no design tables
ERROR 5486: Workspace cannot be NULL
ERROR 5487: Design name cannot be NULL
ERROR 5564: Deployment Parallelism cannot be less than zero
ERROR 5565: Deployment parallelism cannot be NULL
ERROR 5573: Error generating results set
ERROR 5575: Error querying designs table
ERROR 5588: Fenced mode false is not supported for string functions
ERROR 5589: Fenced mode is not supported for SQL functions
ERROR 5591: Hurryup parameter cannot be NULL
ERROR 5597: Invalid input query: 'string'
ERROR 5650: Query without referencing any design tables is not supported: string
ERROR 5657: Segmentation type of the projection string is not supported for encoding design, skipping
ERROR 5694: Weight for query_text 'string' is 'value'. Only positive weight values are accepted
ERROR 5747: analyzeStats flag cannot be NULL
ERROR 5773: Cannot output deployment script because design is not available
ERROR 5792: continueAfterError flag cannot be NULL
ERROR 5817: Deploy flag cannot be NULL
ERROR 5855: Did not find any tables to analyze correlations on
ERROR 5857: dropDesignAndCtx flag cannot be NULL
ERROR 5858: dropProjs flag cannot be NULL
ERROR 5866: Error while analyzing correlations for design table string.string
ERROR 5867: Error while analyzing count distincts for design table string.string
ERROR 5868: Error while analyzing count distincts on correlation sample for design table string.string
ERROR 5869: Error while analyzing segmentation skew for design table string.string
ERROR 5870: Error while dropping existing correlations in design table string
ERROR 5871: Error while loading or analyzing correlations in design tables for design string
ERROR 5907: Force option cannot be NULL
ERROR 5908: forceRecomputation flag cannot be NULL
ERROR 5919: Input cannot be NULL
ERROR 5938: isAdminUser flag cannot be NULL
ERROR 5939: K-safety of incremental designs must match the current system k-safety (which is value)
ERROR 5979: onlyScript flag cannot be NULL
ERROR 5980: outputScript flag cannot be NULL
ERROR 6040: Table string has no correlations
ERROR 6041: Table string has no data. As a result, no correlations were analyzed on this table
ERROR 6042: Table string has no statistics or data. As a result, no correlations were analyzed on this table
ERROR 6043: Table string has no statistics or data. As a result, no correlations were read into this table
ERROR 6049: The mode of analyzing correlations cannot be NULL
ERROR 6050: The mode of analyzing correlations is invalid
ERROR 6096: User has insufficient privileges on table string.string
ERROR 6223: Design K-safety should be in [0,value] range
ERROR 6247: Error during deployment: no projections found for deployment string
ERROR 6489: Error querying v_catalog.projections table
ERROR 7191: Projection with expressions string is not supported for encoding design
ERROR 7192: Projection with expressions string is not supported for encoding design, skipping
ERROR 7379: Cannot generate unique name for Database Designer schema
ERROR 8487: No new projections recommended for deployment by OBD

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
Error Messages Associated with SQLSTATE VP000

This topic lists the errors associated with the SQLSTATE VP000.

SQLSTATE VP000 Description

ERRCODE_USER_PROC_ERROR

Error messages associated with this SQLState

ERROR 2059: string with specified name and parameters does not exist: string
ERROR 2315: Can't have more than one parameters with the same name: string
ERROR 3354: External procedures directory has not been set
ERROR 3355: External procedures have not been installed
ERROR 3465: Function cannot be moved into a system schema
ERROR 4322: Parameter must have names
ERROR 4323: Parameter type is not valid for an external procedure: string
ERROR 4373: Phase value of multi-phase transform function marked prepass
ERROR 4430: Procedures cannot be created in a system schema
ERROR 5232: Unrecognized identifier: string
ERROR 5368: User Defined Aggregates do not support fenced execution mode
ERROR 5372: User Defined Function type not found
ERROR 5374: User Defined Scalar Function string is giving bad numeric precision value, the maximum is value
ERROR 5375: User Defined Scalar Function string is giving bad string typmod value, the minimum is value
ERROR 5376: User Defined Scalar Function string is giving typmod of precision value, larger than the max precision value
ERROR 5377: User Defined Scalar Function string provided non-zero precision (value) for Interval Year To Month
ERROR 5378: User Defined Scalar Function string provided precision value, larger than the maximum precision value
ERROR 5379: User Defined Scalar Function string provided range for Day To Second, but the function's return type is Interval Year To Month
ERROR 5380: User Defined Scalar Function string provided range for Year To Month, but the function's return type is Interval Day To Second
ERROR 5684: User Defined Extension cannot be created in a system schema
ERROR 6051: The schema has been dropped
ERROR 6576: Schema does not exist: string
ERROR 6580: Stemmer UDx string must have VARCHAR or LONG VARCHAR parameter type
ERROR 6581: Stemmer UDx string must have VARCHAR or LONG VARCHAR return type
ERROR 6642: Argument types must be specified for stemmer "string"
ERROR 6869: Multi-phase transform function must have at least one phase
ERROR 6961: Stemmer UDx string must return a single argument
ERROR 7010: Tokenizer required to create text index
ERROR 7015: UDSF or SQL Function with specified name and parameters does not exist: string
ERROR 7082: "string" is an incorrect EarlyMaterialize directive
ERROR 7128: Tokenizer "string" with specified argument types could not be found
ERROR 7129: Tokenizer UDx string must be polymorphic or have a single input field of CHAR, VARCHAR, LONG VARCHAR, or USER DEFINED argument type
ERROR 7139: View cannot be moved into a system schema
ERROR 7582: User Defined Transforms with cursors do not support fenced execution mode

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE VP001

This topic lists the errors associated with the SQLSTATE VP001.

SQLSTATE VP001 Description

ERRCODE_USER_PROC_EXEC_ERROR

Error messages associated with this SQLState

ERROR 2376: Cannot connect to UDx side process (pid = value) during cancel: string
ERROR 2837: Could not create pipe for user procedure execution, errno=value
ERROR 2853: Could not execute user procedure: fork error
ERROR 2858: Could not find function definition
ERROR 2861: Could not find running procedure for string, proc ID=[value]
ERROR 3223: Error in calling string() for User Defined Function string at [string:value], error code: value, message: string
ERROR 3224: Error in calling string() for User Defined Scalar Function string at [string:value], error code: value, message: string
ERROR 3398: Failure in UDx RPC call string() (pid = value): string
ERROR 3399: Failure in UDx RPC call string(): string
ERROR 4424: Procedure execution error: exit status=value
ERROR 4425: Procedure execution error: procedure killed by signal (value)
ERROR 4538: Received message with unexpected type string
ERROR 5170: Unexpected exception from in calling string() for User Defined Scalar Function string
ERROR 5171: Unexpected exception in calling string() in User Defined Function string
ERROR 5205: Unknown error killing procedure string
ERROR 5395: User procedure execution failed
ERROR 5398: User-defined Analytic Function string produced fewer output rows than input rows
ERROR 5399: User-defined Scalar Function string outputted a timezone (value) not in allowed range (value, value)
ERROR 5400: User-defined Scalar Function string produced fewer output rows (value) than input rows (value)
ERROR 5430: Vertica process is not allowed to kill procedure string
ERROR 5580: Failure sending parameters block because the value parameters require value bytes, which exceeds the maximum size of value bytes
ERROR 5604: Invalid procedure file: [string]
ERROR 5638: Procedure file [string] cannot be owned by root
ERROR 5639: Procedure file [string] must be executable by vertica user (dbAdmin)
ERROR 5640: Procedure file [string] must be owned by specified procedure user
ERROR 5641: Procedure file [string] must enable set UID attribute
ERROR 5656: Root cannot execute external procedure
ERROR 5683: User 'string' not found on node
ERROR 5861: Error calling string() in User Function string at [string:value], error code: value, message: string
ERROR 5863: Error during setting up function, message: string
ERROR 6085: Unexpected exception calling string() User Function in string
ERROR 6086: Unexpected exception calling destroyUDxFenced()
ERROR 6087: Unexpected exception thrown by UDFileSystem at [string:value], error code: value, message: string
ERROR 6668: Can’t access [string]: No filesystem is mapped to scheme string
ERROR 6706: Could not find filesystem for scheme string
ERROR 6713: Couldn’t cancel the user procedure, string
ERROR 6756: Error in calling destructor for UDFilter function at [string:value], error code: value, message: string
ERROR 6757: Error in calling destructor for UDParse function at [string:value], error code: value, message: string
ERROR 6758: Error in calling destructor for UDSource function at [string:value], error code: value, message: string
ERROR 6759: Error in calling ~string() for User Defined Function string at [string:value], error code: value, message: string
ERROR 6760: Error in calling ~string() for User Defined Scalar Function string at [string:value], error code: value, message: string
ERROR 6783: Filesystem does not support glob string
ERROR 6859: makeConnection: send_msg(value) did not succeed: string
ERROR 7025: Unexpected exception in calling destructor in UDFilter function
ERROR 7026: Unexpected exception in calling destructor in UDParse function
ERROR 7027: Unexpected exception in calling ~string() for User Defined Scalar Function string
ERROR 7028: Unexpected exception in calling ~string() in User Defined Function string
ERROR 7097: Failure in UDx RPC call string()
ERROR 7112: Procedure reported: string
ERROR 7685: User-defined Scalar Function string produced more output rows (value) than input rows (value)
ERROR 7712: Error during cleanup for User Defined Function string: string
ERROR 7981: Cannot reserve memory from the JVM resource pool
ERROR 8092: Failure in UDx RPC call string() in User Defined Object [string]: string
ERROR 8468: Error calling string() in User Function string, message: string
ERROR 8486: Invalid S3 path specified [string]

Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.

Error Messages Associated with SQLSTATE VX001

This topic lists the errors associated with the SQLSTATE VX001.

SQLSTATE VX001 Description

ERRCODE_INTERNAL_ERROR

Error messages associated with this SQLState

ERROR 2025: string Error occurred during BZIP initialization. BZIP error code: value
ERROR 2027: string Error occurred during ZLIB initialization. ZLIB error code: value, Message: string
ERROR 2111: Active queue cleared while running
ERROR 2405: Cannot do boundary analysis on type value
ERROR 2616: Cluster recovery failed, try again
ERROR 2928: Couldn’t check this session’s state
ERROR 3099: Did not find a variable
ERROR 3198: Error dropping table partition, data in WOS
ERROR 3211: Error during recovery running string queries, cannot continue: string (at string:value)
ERROR 3212: Error during recovery running string: string (at string:value)
ERROR 3220: Error generating query for: string
ERROR 3245: Error parsing string
ERROR 3292: Event apply failed
ERROR 3302: Exception decoding the response we just locally encoded
ERROR 3483: Got unexpected error code from spread: value, string
ERROR 3818: JOIN qualifications to not refer to the correct relation(s)
ERROR 3969: More than one variable found
ERROR 4372: pg_analyze_and_rewrite for View query failed
ERROR 4514: Raw parse of View query string failed
ERROR 4545: Recovery Error: Cannot get projections on local node
ERROR 5236: unrecognized node type value
ERROR 5237: unrecognized node type value in postprocess conditions
ERROR 5526: Already have a ready_recv string, ignoring
ERROR 5539: Cannot find buddy projection's statistics for collecting row counts, min and max
ERROR 5540: Cannot find buddy projections for collecting row counts, min and max
ERROR 5541: Cannot find the up-nodes of buddy projection for collecting row counts, min and max
ERROR 5679: Unrecognized order by expression
ERROR 5680: Unrecognized select column list
ERROR 5695: With query is not a Select Statement
ERROR 5719: Path Sampling failed. Try a different random seed for the pathSampling hint
ERROR 5802: Could not stop all dirty transactions[txnId = string]
ERROR 5865: Error while analyzing approximate count distincts on table string.string
ERROR 6062: Too Many User defined types
ERROR 6248: Error occurred during LZO decompression: LZO checksum error on compressed data, possibly due to file corruption
ERROR 6249: Error occurred during LZO decompression: LZO checksum error on uncompressed data, possibly due to file corruption
ERROR 6250: Error occurred during LZO decompression: LZO expected destination length larger than BLOCK_SIZE, possibly need to recompile lzop, or set --blocksize to a larger value
ERROR 6251: Error occurred during LZO decompression: LZO expected destination length larger than MAX_BLOCK_SIZE, possibly due to file corruption
ERROR 6252: Error occurred during LZO decompression: LZO expected source length is wrong, possibly due to file corruption
ERROR 6253: Error occurred during LZO header processing: expecting more than value bytes, possibly file corrupted
ERROR 6254: Error occurred during LZO header processing: return code value, possibly due to file corruption
ERROR 6428: The password for "string", encryption algorithm string does not match the effective server configured encryption algorithm string, please expire the password to reset
ERROR 6429: The sending password for "string", encryption algorithm string does not match the effective server configured encryption algorithm string
ERROR 6440: Trying to change password for "string", but password encryption algorithm does not match, server configured string, client send in string
ERROR 6468: Wrong Password Security Algorithm string
ERROR 6482: Failed to parse Access Policies for table "string" [string]
ERROR 6678: Cannot extract relations in the query
ERROR 6738: DML is running while collecting dirty txns
ERROR 6768: Error retrieving Group ROS [value] of ROS [value]
ERROR 6771: Fail to get table recovery status when node is not INITIALIZING/RECOVERING/READY/UP
ERROR 6775: Failed to generate an annotated query: string
ERROR 6779: Failed to recover all tables, would retry!
ERROR 6780: Failed to recover node, shutting down...
ERROR 6820: Input query cannot be deparsed
ERROR 6821: Input query is not supported
ERROR 6893: Output annotated query is not supported
ERROR 6895: Output stream failed to initialize
ERROR 6927: Query not ready to write to export file
ERROR 6930: Recovery Error: Cannot get projections of table string
ERROR 6931: Recovery Error: Cannot get projections of tables having prejoin projections
ERROR 6942: ROS [value] is in a bundle without a storageId
ERROR 6995: The content of the input query saved previously changed
ERROR 7032: Unexpected segmentation for constraint projection
ERROR 7274: Recovery Error: projection recovery start epoch is behind AHM in string phase. Has AHM been advanced during recovery?
ERROR 7348: value projections out of value projections fail to moveout to epoch value
ERROR 7493: OpenSSL RAND_bytes returns error; value
ERROR 7535: ROS value starts at epoch value, end at epoch value, straddle truncate epoch value
ERROR 7645: Configured password type string not admissable under FIPS
ERROR 7656: Effective password type string not admissable under FIPS
ERROR 7661: MDS not permitted in FIPS mode
ERROR 7668: The server password string is not allowed on FIPS systems; please have DB admin correct this
ERROR 7669: Trying to change password for "string", but MDS hash algorithm not permitted
ERROR 7855: Found SAL corruption
ERROR 7873: Property string is in string, not string
ERROR 7874: Property string not found in string
ERROR 7944: Cannot set SET-USING because refresh_columns failed
ERROR 8150: Session is running a transaction even though Initiator is not. Has transaction value folded without rollback or commit? Call description: string
ERROR 8387: Cluster membership change interfered with an operation to initialize tables for recovering node string
ERROR 8404: Cannot initialize necessary map for recovery
ERROR 8485: Input query is not supported for EXPLAIN ANNOTATED
ERROR 8492: Uploader caught unknown exception: string
Error Messages Associated with SQLSTATE VX002

This topic lists the errors associated with the SQLSTATE VX002.

SQLSTATE VX002 Description

ERRCODE_DATA_CORRUPTED

Error messages associated with this SQLState

ERROR 2940: CRC Check Failure Details:
  File Name: string
  File Offset: value
  Compressed size in file: value
  Memory Address of Read Buffer: value
  Pointer to Compressed Data: value
  Memory Contents:
    string

ERROR 3030: Delete: could not find a data row to delete (data integrity violation?)

ERROR 3218: Error finalizing ROS DataTarget

ERROR 3219: Error flushing data file [string]

ERROR 3409: FileColumnReader: block string @ value 's CRC value doesn't match record value

ERROR 3410: FileColumnReader: Decompression error in string at offset value

ERROR 4762: Sort Order Violation:
  Row Position: value
  Column Index: value
  Last Row: string
  This Row: string

ERROR 5704: Delete (Join): could not find a data row to delete (data integrity violation?)

ERROR 6767: Error reading from orc parser input stream [string]: file shorter than expected, read to value, requested to value

ERROR 7581: Unsupported model_type

ERROR 7767: Error reading from parquet parser input stream [string]: file shorter than expected, read to value, requested to value

ERROR 7859: Block[value] has unknown type. (size: value, type: value, pad: value, count: value, position: value)

ERROR 7860: Block[value]->count is not consistent with other blocks. (value != value)
Error Messages Associated with SQLSTATE VX003

This topic lists the errors associated with the SQLSTATE VX003.

SQLSTATE VX003 Description

ERRCODE_INDEX_CORRUPTED

Error messages associated with this SQLState

ERROR 3544: Index corruption. string: string

Error Messages Associated with SQLSTATE VX004

This topic lists the errors associated with the SQLSTATE VX004.

SQLSTATE VX004 Description

ERRCODE_PLAN_TO_SQL_INTERNAL_ERROR

Error messages associated with this SQLState

ERROR 6890: Optimizer-generated annotated query has unexpected error. Please report to HPE Vertica
Note: The Vertica User Community contains knowledge base articles, blogs, and forum posts that may help you resolve these errors.
The Vertica Glossary defines terms that are common and specific to Vertica.
### Access Rank

Determines the speed at which a column can be accessed. Columns are stored on disk from the highest ranking to the lowest ranking in which the highest ranking columns are placed on the fastest disks and the lowest ranking columns are placed on the slowest disks.
Administration Host

The host on which the Vertica rpm package was manually installed. Always run the Administration Tools on this host if possible.
Administration Tools

One of the ways you can manage a Vertica database is provided in the form of a graphical user interface, called Administration Tools.

This tool allows you to perform various tasks quickly and easily, some of which are:

- View the state of the database cluster
- Create a database
- Start a database
- Stop a database
- Run Database Designer
- Connect to a database using vsq1

Using the following command, always run the Administration Tools on the Administration Host if possible.

$ /opt/vertica/bin/adminTools

Note: Throughout the Vertica documentation, you might see Administration Tools referred to as Admin Tools or admintools or adminTools. They all refer to the same utility.

For more information, see Administration Tools.
Agent

A daemon process that runs on each Vertica cluster node. The agent is used by certain clients, such as Management Console, to administer Vertica.

Agents monitor Vertica database clusters and communicate with their clients to provide the following functionality:

- Provide local access, command, and control over database instances on a given node, using functionality similar to Administration Tools
- Report log-level data from the Administration Tools and Vertica log files
- Cache details from long-running jobs—such as create/start/stop database operations—that you can view through your browser
- Track changes to data-collection and monitoring utilities and communicate updates to clients
- Specifically for MC, communicate between all cluster nodes and MC through a webhook subscription, which automates information sharing and reports on cluster-specific issues like node state, alerts, events, and so on

The agent runs on port 5444, which must be accessible to agent clients.
Anchor Table

Database table that is the source for data in a projection.
Ancient History Mark (AHM)

Also known as AHM, the ancient history mark is the oldest epoch whose data is accessible to historical queries. Any data that precedes the AHM is eligible to be purged.

For detailed information, see Understanding Vertica Epochs in the Vertica Knowledgebase.
Apportioned Load

An *apportioned load* is a divisible load, such that you can load a single data file on more than one node. Each load starts at a different offset, requiring a parser that supports apportioning. Some of the parsers built into Vertica support apportioned load. Using the SDK, you can write parsers that perform apportioned loads.
Authentication Service (AS)

A service that usually runs on the same host as the Kerberos Key Distribution Center (KDC). The AS issues tickets for a requested service. The tickets are in turn given to users for access to the service. The AS answers requests from clients that do not send credentials with a request. The AS is generally used to gain access to the ticket-granting service (TGS) by issuing a ticket-granting ticket (TGT).
Automatic Tuple Mover (ATM)

The Automatic Tuple Mover (ATM) refers to the Tuple Mover operations that occur automatically at regular intervals set by configuration parameters. For example, two configuration parameters control the TM mergeout and moveout intervals, which are both ATM activities. Conversely, invoking the `do_tm_task` function is not an ATM activity. For information about changing the TM configuration parameters, see Tuple Mover Parameters in the Administrator's Guide for further information.
Bitstring

A sequence of bits.
Buddy Projection

Required for K-safety. Two projections are considered to be buddies if they contain the same columns and have the same hash segmentation, using different node ordering.

For more information, see:

- High Availability With Projections
- Designing Segmented Projections for K-Safety
- GET_PROJECTION_STATUS
Bulk Loading

A process of loading large amounts of data, such as an initial load of historic data.
**C-Store**

A research project at MIT, Brandeis, Brown, and the University of Massachusetts (Boston), on which Vertica is based.
Cardinality

Refers to the number of unique values for a given column in a relational table:

- **High cardinality**: Refers to columns containing values that are highly unique, such as a customer ID or an employee e-mail address. For example, in the Vertica VMart schema, the employee_dimension table contains an employee_key column. This column contains values that uniquely identify each employee. Since the values in this column are unique and could be numerous, the column's cardinality type is referred to as high cardinality.

- **Normal cardinality**: Refers to columns containing values that are less unique, such as job titles and street addresses. An example of a normal-cardinality column would be job_title or employee_first_name in the employee_dimension table, where many employees could share the same job title or same first name.

- **Low cardinality**: Refers to a low number of unique values, relative to the overall number of records in a table. For example, in the employee_dimension table, the column called employee_gender would contain two unique values: 'Male' or 'Female'. Since there are only two values possible in this column, cardinality is low.
Catalog

A set of files that contains information (metadata) about the objects—such as nodes, tables, constraints, and projections—in a database. Vertica maintains a catalog on each node in the cluster.
Catalog Path

A storage location used to store the database catalog.
**Checkpoint**

Every time the Tuple Mover performs a move-out operation for a given projection, it records a Checkpoint Epoch for that projection, representing an epoch up to which the projection has no data in the WOS. The minimum checkpoint epoch across all projections on that node is called the node’s Checkpoint Epoch. It represents a point in time up to which all the data was moved out to disk. If a K-safe=1 database experiences a single-node failure, the node's recovery process attempts to rebuild the data beyond the Checkpoint Epochs from other nodes. If all nodes fail, such as during a power outage, Vertica recovers the database back to the minimum Checkpoint Epochs across all the nodes, known as the Last Good Epoch (LGE).
Cluster

The concept of Cluster in the Vertica Analytics Platform is a collection of hosts with the Vertica software packages (RPM or DEB) that are in one admin tools domain. You can access and manage a cluster from one admintools initiator host.
Columnar Tables

Vertica database tables consisting of structured data columns. The term differentiates these tables from Flex (or Flexible) tables, which minimally contain one column of unstructured, or semi-structured data. Flex tables can also have structured data columns, but they are not required. Compare with Flexible Tables.
Control node

In a large-cluster arrangement, Vertica Analytic Database delegates control message responsibilities to a subset of nodes, called control nodes, to improve control message performance. Control nodes communicate with each other, and non-control nodes are assigned to a control node for control message communications. See Large Cluster in the Administrator's Guide.
Correlated columns

Two columns are correlated if the value of one column is related to the value of the other column. For example, state name and country name columns are strongly correlated because the city name usually, but perhaps not always, identifies the state name. The city of Conshohoken is uniquely associated with Pennsylvania, whereas the city of Boston exists in Georgia, Indiana, Kentucky, New York, Virginia, and Massachusetts. In this case, city name is strongly correlated with state name.
Critical Node

A critical node is a node whose failure would cause the database to become unsafe and force a shutdown. Nodes can become critical for the following reasons:

- A node has the only copy of a particular projection.
- Fewer than half of your nodes are active.

The `V_MONITOR.CRITICAL_NODES` system table lists the critical nodes, if any, in your cluster.
**Current epoch**

The epoch into which data (COPY, INSERT, UPDATE, and DELETE operations) is currently being written.
Data Collector

A utility that collects and retains database monitoring information.

Note: Data Collector works in conjunction with an advisor tool called Workload Analyzer (WLA), which intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history.

Data Collector retains history of important system activities and records essential performance and resource utilization counters. You can use information retained by the Data Collector in the following ways:

- As a reference for what actions users have taken
- To locate performance bottlenecks
- To identify potential improvements to Vertica configuration

Data Collector is on by default. If you need to change this behavior, see Data Collector Parameters in the Administrator’s Guide. See also the SQL Reference Manual for information about data collection control functions.
**Data Path**

A storage location that contains actual database data files.
Database

A cluster of nodes that, when active, can perform distributed data storage and SQL statement execution through administrative, interactive, and programmatic user interfaces.
**Database Designer**

A tool that analyzes a logical schema definition, sample queries, and sample data, and creates a physical schema (projections) in the form of a SQL script that you deploy automatically or manually. The script creates a minimal set of superprojections to ensure K-safety, and, optionally, non-superprojections. In most cases, the projections created by the Database Designer provide excellent query performance within physical constraints.

The Database Designer can create two distinct design types. The design you choose depends on what you are trying to accomplish:

- **Comprehensive Design**

- **Incremental Design**

You can also create custom designs if the Database Designer does not meet your needs.

For detailed information, see Creating a Database Design.
**Database superuser**

The automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization, or any user that has been granted the PSEUDOSUPERUSER role. Do not confuse the concept of a database superuser with Linux superuser (root) privilege. A database superuser cannot have Linux superuser privilege.
DELTAVAL (delta encoding)
Stores only the differences between sequential data values instead of the values themselves.
Derived Column

A column whose values are calculated by an expression at load time. The expression is specified within the COPY statement, and the column exists in the target database.
**Dimension Table**

Sometimes called a lookup or reference table, a dimension table is one of a set of companion tables to a large (fact/anchor) table in a star schema. It contains the PRIMARY KEY column corresponding to the join columns in fact tables. For example, a business might use a dimension table to contain item codes and descriptions.

Dimension tables can be connected to other dimension tables to form a hierarchy of dimensions in a snowflake schema.
Directed Query

A saved set of instructions that direct the optimizer to generate a query plan for a given query. The query plan consists of SQL annotated with hints. A directed query pairs two components:

- **Input query**: A query that triggers use of this directed query when it is active.
- **Annotated query**: A SQL statement with embedded optimizer hints. The annotated query is used by the optimizer in creating a query plan for the specified input query.
Encoding

The process of converting data into a standard format. In Vertica, encoded data can be processed directly, while compressed data cannot. Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order.
Epoch

A logical unit of time in which a single change is made to data in the system.
**Epoch Map**

A catalog object that provides mapping between time and epochs. Specifically, an epoch map contains a list of epoch numbers and their associated timestamps.
**Executor Node**

Any node that participates in executing a specific SQL statement. The initiator node can, and usually does, also function as an executor node.
External Procedure

A procedure external to Vertica that you create, maintain, and store on the server.
Event Series

Tables with a time column, most typically a timestamp data type.
Flexible Tables

Vertica database tables that minimally contain two columns:

__identity__: A real column with an incrementing IDENTITY value for partitioning and sorting. Used if no other columns serve this purpose.

__raw__: A real LONG VARBINARY column containing unstructured, or semi-structured data.

You can create Flex tables with additional real columns, but they are not required. Compare with Columnar Tables.
Flattening (subqueries and views)

Occurs when a subquery or named view is internally rewritten so the subquery is combined with the outer query block. The result sets of the original and flattened queries are exactly the same, but the flattened query usually benefits from significant performance improvements.
**Full Backup**

Consists of copying each catalog and all data files (ROS containers) on each node, as well as the complete `/opt/vertica/config` directory.
**GENERAL Pool**

A special built-in pool that represents the total amount of RAM available to the resource manager for use by queries. Other pools can borrow memory from the GENERAL pool. See also Built-In Pools.
**Grant**

Vertica defines GRANT in two ways:

1. Grant a user privileges to access database objects using any GRANT statement, except GRANT (Authentication).

2. Associate a user-defined authentication method with a user through GRANT (Authentication). This operation differs from GRANT <privileges> as authentication methods are “associated” with a user or role and privileges are “granted” to a user or role.

**Examples**

**Grant Access Privileges to a User**

This example shows how to grant a user, Joe, privileges to access the online_sales schema:

```sql
=> GRANT USAGE ON SCHEMA online_sales TO Joe;
```

**Associate an Authentication Method with a User**

This example shows how to associate the v_ldap authentication method to user jsmith:

```sql
=> GRANT AUTHENTICATION v_ldap TO jsmith;
```
Grouped ROS

A grouped ROS is a highly-optimized, read-oriented physical storage structure organized by projection. A grouped ROS makes heavy use of compression and indexing. Unlike a ROS, a grouped ROS stores data for two or more grouped columns in one disk file.
Hash Segmentation

Specifies how to segment projection data for distribution across some or all cluster nodes. You can specify segmentation for a table and a projection. If a table definition specifies segmentation, Vertica uses it for that table's auto-projections.

It is strongly recommended that you use Vertica's built-in HASH function, which distributes data evenly across the cluster, and facilitates optimal query execution.

For more detailed information, see Projection Segmentation.
Historical Data

Refers to any data in memory or physical storage other than the current epoch. Historical data includes all COPY, INSERT, UPDATE, and DELETE operations, including deleted rows. This allows Vertica to run a historical query on data written up to and including the epoch representing the specified date and time.
Historical Query

A query that retrieves data from a snapshot of the database, taken at a specific timestamp or epoch. For details, see Historical Queries in Analyzing Data.
Immutable (Invariant) Functions

When run with a given set of arguments, immutable functions such as \( \text{AVG}( ) \) always produce the same result, regardless of environment or session settings such as locale.

Some immutable functions can take an optional stable argument; in this case they are treated as Stable functions.
**Initiator Node**

In the context of a client connection, the initiator node is the node associated with the specific host to which the connection was made. The initiator node can, and usually does, also function as an executor node, and is generally where the most descriptive log messages reside.
K-Safety

K-safety sets fault tolerance for the database cluster, where K can be set to 0, 1, or 2. The value of K specifies how many copies Vertica creates of segmented projection data. If K-safety for a database is set to 1 or 2, Vertica creates K+1 instances, or buddies, of each projection segment. Vertica distributes these buddies across the database cluster, such that projection data is protected in the event of node failure. If any node fails, the database can continue to process queries so long as buddies of data on the failed node remain available elsewhere on the cluster.

For more information, see Using Database Designer.
**Last Epoch**

The last epoch is the current epoch minus one. SELECT statements under READ COMMITTED isolation read from the last epoch.
**Last Good Epoch (LGE)**

A term used in manual recovery, LGE (Last Good Epoch) refers to the most recent epoch that can be recovered.
**Live Aggregate Projection**

A live aggregate projection contains columns with values that are aggregated from columns in its anchor table. When you load data into the table, Vertica aggregates the data before loading it into the live aggregate projection. On subsequent loads—for example, through `INSERT` or `COPY`—Vertica recalculates aggregations with the new data and updates the projection.
Locale

Locale specifies the user's language, country, and any special variant preferences, such as collation. Vertica uses locale to determine the behavior of certain string functions. Locale also determines the collation for various SQL commands that require ordering and comparison, such as aggregate GROUP BY and ORDER BY clauses, joins, and the analytic ORDER BY clause.

The default locale for a Vertica database is en_US@collation=binary (English US). You can define a new default locale that is used for all sessions on the database. You can also override the locale for individual sessions. However, projections are always collated using the default en_US@collation=binary collation, regardless of the session collation. Any locale-specific collation is applied at query time.

If you set the locale to null, Vertica sets the locale to en_US_POSIX. You can set the locale back to the default locale and collation by issuing the vsql meta-command \locale. For example:

```bash
=> set locale to ''; INFO 2567: Canonical locale: 'en_US_POSIX'
Standard collation: 'LEN'
English (United States, Computer)
SET
=> \locale en_US@collation=binary;
INFO 2567: Canonical locale: 'en_US'
Standard collation: 'LEN_KBINARY'
English (United States)
=> \locale
en_US@collation-binary;
```

You can set locale through ODBC, JDBC, and ADO.net.
**Logical Schema**

Consists of a set of tables and referential integrity constraints in a Vertica database. The objects in the logical schema are visible to SQL users. The logical schema does not include projections, which make up the physical schema.
Location Label

A label assigned to a storage location. Location labels identify the location so you can create object storage policies. The labeled location you use in a storage policy becomes the default storage location for the object.
Management Console

A database management tool that provides a unified view of your Vertica database and lets you monitor multiple clusters from a single point of access. See Management Console in Vertica Concepts.
MC Super (superuser administrator)

Called Super on the MC interface, the MC super is the Linux user account that gets created when you configure MC. See Configuring MC and SUPER Role (mc).
MC-managed Database

A Vertica database that an MC SUPER or ADMIN user creates on MC or imports into the MC interface, along with the database cluster. When MC users are granted database privileges, their privileges are defined through the MC itself and pertain only to databases managed on the MC. See About MC Privileges and Roles.
Manual Recovery

The process of recovery after an unclean shutdown of the database, where the administrator must accept recovery from the Last Good Epoch, which could lead to loss of some recently loaded data. See Failure Recovery.
Mergeout

Mergeout is the Tuple Mover process that consolidates ROS containers and purges deleted records.
Meta-Functions

Used to query or change the internal state of Vertica and are not part of the SQL standard. See Vertica Meta-functions in the SQL Reference Manual.
**Metadata**

Data that describes the data in a database, such as data type, compression, constraints, and so forth, for each column in the tables. Metadata is stored in the Vertica catalog.
**Moveout**

The tuple mover operation of moving data from the WOS (Write Optimized Store), which is kept in memory, to the ROS (Read Optimized Store), which is kept on disk.
Node

A host configured to run an instance of Vertica. It is a member of the database cluster. For a database to have the ability to recover from the failure of a node requires a database K-safety value of at least 1 (3+ nodes).
Non-Repeatable Read

Occurs in a READ COMMITTED isolation level when two identical queries in the same transaction produce different results. This occurs when another transaction commits changes that alter the results of the query after the first query has completed and before the second query has begun.
**NUL**

Represents a character whose ASCII/Unicode code is zero, sometimes qualified "ASCII NUL".
**NULL**

Means *no value*, and is true of a field (column) or constant but not of a character.
OID

An object identifier (OID) is an identifier used to name an object. Structurally, an OID consists of a node in a hierarchically-assigned namespace, formally defined using the International Telecommunication Union-Telecommunication's (ITU-T) Abstract Syntax Notation standard (ASN.1). Vertica database objects are always assigned an OID. The OID can be used directly in some statements, such as HAS_TABLE_PRIVILEGE.

Parallel Load

*Parallel load* is the process of loading data on any available node in the cluster (not necessarily the local node). Use this approach in combination with wildcards (such as *.dat) to load data files in parallel in a distributed manner. See COPY ON ANY NODE in COPY Parameters.
Path (quality plan)

The execution strategy of the Vertica cost-based query optimizer, denoting a sub operation in the query plan.
Partitioning

Specifies how data is organized within individual nodes. Partitioning attempts to introduce hot spots within the node, providing a convenient way to drop data and reclaim the disk space.

Note: Partitioning and segmentation are terms often used interchangeably for other databases, but they have completely separate goals in Vertica regarding data localization. See Partitioning and Segmentation in the Administrator's Guide for details.
**Phantom Read**

Occurs in a READ COMMITTED isolation level when two identical queries in the same transaction produce different collections of rows. This occurs because a table lock is not acquired on SELECT during the initial query.
Physical Schema

Consists of a set of projections used to store data on disk. The projections in the physical schema are based on the objects in the logical schema.
Physical Schema Design

A usable K=1 design based on an analysis of the sample data files and queries (if available). The physical schema design contains segmented (fact table) superprojections and replicated (dimension table) superprojections.
Primary Column

A column that is loaded from raw data, and not derived from an expression.
Projection

Optimized collections of table columns that provide physical storage for data. A projection can contain some or all of the columns of one or more tables.

For conceptual information about Vertica projections, see Physical Schema in Vertica Concepts. For information about using and managing projections, see Working with Projections in the Administrator's Guide.
**Projection set**

A group of buddy projections that are safe for a given level of K-safety. When K=1, there are two buddies in a set; when K=2, there are three buddies. The Database Designer assigns all projections in a projection set the same base name so they can be identified as a group.

A projection must be part of a projection set before it is refreshed. Once a projection set is created (by creating buddies), the set is refreshed in a single transaction.
Query Cluster Level

Determines the number of sets used to group similar queries. The query cluster level can be any integer from one (1) to the number of queries to be included in the physical schema design.

Queries are generally grouped based on the columns they access and the way in which they are used. The following work loads typically use different types of queries and are placed in different query clusters: drill downs, large aggregations, and large joins. For example, if a reporting tool and dashboard both access the same database, the reporting tool is likely to use a drill down to access a subset of data and the dashboard is likely to use a large aggregation to look across a large range of data. In this case, there would be at least two (2) query clusters.
**Query Optimizer**

The component that evaluates different strategies for running a query and picks the best one.
Recovery

Vertica can restore the database to a fully functional state after one or more nodes in the system experiences a software- or hardware-related failure. Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure can cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, queries other nodes in the cluster to recover lost objects and catch up with database changes.
**Referential Integrity**

Consists of a set of constraints (logical schema objects) that define primary key and foreign key columns.

- Each small table must have a PRIMARY KEY constraint.

- The large table must contain columns that can be used to join the large table to smaller tables.

- Outer join queries produce the same results as the corresponding inner join query if there is a FOREIGN KEY constraint on the outer table. Note that the inner table of the outer join query must always have a PRIMARY KEY constraint on its join columns.
**Refresh (projections)**

Ensures that all projections on a node are up-to-date (can participate in query execution). This process could take a long time, depending on how much data is in the table(s).

For more information, see *Refreshing Projections*. 
Resegmentation

A process that Vertica performs automatically during query execution that distributes the rows of an existing projection or intermediate relation evenly to each node in the cluster. At the end of resegmentation, every row from the input relation is on exactly one node. Vertica resegments data when the input does not have the segmentation required to compute the requested result efficiently and correctly.
Resource Pool

A resource pool comprises a pre-allocated subset of the system resources, with an associated queue. A resource pool is created using the CREATE RESOURCE POOL command as described in the SQL Reference Manual.
Resource Manager

In a single-user environment, the system can devote all resources to a single query and get the most efficient execution for that one query. However, in an environment where several concurrent queries are expected to run at once, there is tension between providing each query the maximum amount of resources (thereby getting fastest run time for that query) and serving multiple queries simultaneously with a reasonable run time. The Resource Manager (RM) provides options and controls for resolving this tension, while ensuring that every query eventually gets serviced and that true system limits are respected at all times.
Role

A role groups together a set of privileges that can be assigned to a user or another role. You can use roles to quickly grant or revoke privileges on multiple tables, schemas, functions or other database entities to one or more users with a single command.

Vertica's implementation of roles conforms to the SQL Standard T331 for basic roles.
Rollback

Transaction rollbacks restore a database to an earlier state by discarding changes made by that transaction. Statement-level rollbacks discard only the changes initiated by the reverted statements. Transaction-level rollbacks discard all changes made by the transaction.

With a ROLLBACK statement, you can explicitly roll back to a named savepoint within the transaction, or discard the entire transaction. Vertica can also initiate automatic rollbacks in two cases:

- An individual statement returns an ERROR message. In this case, Vertica rolls back the statement.
- DDL errors, systemic failures, dead locks, and resource constraints return a ROLLBACK message. In this case, Vertica rolls back the entire transaction.

Explicit and automatic rollbacks always release any locks that the transaction holds.
**ROS (Read Optimized Store)**

Read Optimized Store (ROS) is a highly optimized, read-oriented, disk storage structure, organized by projection. ROS makes heavy use of compression and indexing.

The Tuple Mover is the Vertica database optimizer component that moves data from memory (WOS) to disk (ROS). The Tuple Mover runs in the background, performing tasks automatically at time intervals determined by its configuration parameters.

For more information, see [Loading Batches Directly into ROS](#).
ROS Container

A ROS (Read Optimized Store) container is a set of rows stored in a particular group of files. ROS containers are created by operations like Moveout or COPY DIRECT. You can query the STORAGE_CONTAINERS system table to see ROS containers. The ROS container layout can differ across nodes due to data variance. Segmentation can deliver more rows to one node than another. Two data loads could fit in the WOS on one node and spill on another.
Savepoint

A savepoint is a special marker inside a transaction that allows commands that execute after the savepoint to be rolled back. The transaction is restored to the state that preceded the savepoint.

Vertica supports two types of savepoints:

- **An implicit savepoint** is automatically established after each successful command within a transaction. This savepoint is used to roll back the next statement if it returns an error. A transaction maintains one implicit savepoint, which it rolls forward with each successful command. Implicit savepoints are available to Vertica only and cannot be referenced directly.

- **Named savepoints** are labeled markers within a transaction that you set through `SAVEPOINT` statements. A named savepoint can later be referenced in the same transaction through `RELEASE SAVEPOINT`, which destroys it, and `ROLLBACK TO SAVEPOINT`, which rolls back all operations that followed the savepoint. Named savepoints can be especially useful in nested transactions: a nested transaction that begins with a savepoint can be rolled back entirely, if necessary.
**Segmentation**

Defines how physical data storage (projections) is stored in a database cluster using the `CREATE PROJECTION` statement. The goal is to distribute data evenly across multiple nodes in the database so that all nodes can participate in query execution.

*Note: Partitioning and segmentation are terms often used interchangeably for other databases, but they have completely separate goals in Vertica regarding data localization. See [Partitioning and Segmentation](#) in the Administrator's Guide for details.*
Session

An occurrence of a user interacting with a database through the use of SQL statements. You can start a session using vsq1 or a JDBC application. In Vertica, the scope of a session is the same as that of a connection. Connecting to the database starts a session, and exiting ends it.

Session-scoped data is preserved beyond the lifetime of a single transaction. Terminating a session truncates a table and deletes all rows.
Snapshot Isolation

An historical query that gets data from the latest epoch (AT LATEST EPOCH). For details, see Historical Queries.
**Spread**

An open source toolkit used in Vertica to provide a high performance messaging service that is resilient to network faults. Spread daemons start automatically when a database starts up for the first time, and the spread process runs on control nodes in the cluster.
**Sort-Merge Join**

If both inputs are pre-sorted, merge joins do not have to do any pre-processing. Vertica uses the term sort-merge join to refer to the case when one of the inputs must be sorted prior to the merge join. Vertica sorts the inner input side but only if the outer input side is already sorted on the join keys.
Stable functions

When run with a given set of arguments, stable functions produce the same result within a single query or scan operation. However, a stable function can produce different results when issued under different environments or at different times, such as change of locale and time zone—for example, SYSDATE() and 'today'.

See also Immutable (Invariant) Functions.
Standalone Resource Pool

A pool that is configured such that it cannot borrow memory from the GENERAL pool. A standalone pool has MEMORYSIZE = MAXMEMORYSIZE.
**Storage Location**

A directory path used by Vertica to store catalog, actual data files, and temporary data files. You can also create storage locations for users, and then grant one or more users access to the storage. You can also create a storage location with a location label, for use in storage policies.
Storage Policy

A database object you create to associate a labeled location as the default storage location for the object. Database object can be a database, schema, table, or min- and max- ranges of a partition.
**Strict**

Indicates that a function always returns null when any of its input arguments is null.
Superprojection

A projection that includes all columns in an anchor table. Vertica uses superprojections to ensure support for all queries and other DML operations.

Under certain conditions, Vertica automatically creates a table's superprojection immediately on table creation. Vertica also creates a superprojection when you first load data into that table, if none already exists. CREATE PROJECTION can create a superprojection if it specifies to include all table columns. A table can have multiple superprojections.

For more information, see How Projections are Created.
Temp Location

A storage location used as temp space by Vertica.
**Temp Space**

Disk space temporarily occupied by temporary files created by certain query execution operations, such as hash joins and sorts, in the case when they have to spill to disk. Such operations might also be encountered during queries, recovery, refreshing projections, and so on. If a temp location is provided to the database, Vertica uses it as temp space.
**Top-K Projection**

A projection that configures the projection data for a Top-K query. A Top-K query is one that retrieves the top \( k \) rows from a group of tuples.

A Top-K projection is a type of live aggregate projection.
Transaction

One or more operations that are executed as a unit of work. At the user level, transactions occur in the current session by a user or script running one or more SQL statements. When you commit a transaction, any changes you make to data in tables using INSERT, DELETE, UPDATE, MERGE, and COPY during the transaction become permanent. If you roll back the transaction, all changes made to table data are undone.

Vertica supports Atomicity, Consistency, Isolation, and Durability (ACID) for SQL transactions.
**Tuple Mover (TM)**

The Tuple Mover manages WOS and ROS data storage. It performs two operations:

- **Moveout** moves data from WOS to ROS. During moveout operations, the Tuple Mover also enforces storage policies for the storage location.

- **Mergeout** combines small ROS containers into larger ones and purges deleted data.

The Tuple Mover automatically performs these tasks in the background, at intervals that are set by its configuration parameters.

For detailed information, see [Tuple Mover Operations](#).
UDx

User-Defined Extensions (UDx) are functions contained in external shared libraries that are developed in C++, Python, Java, or R using the Vertica SDK. The external libraries are defined in the Vertica catalog using the CREATE LIBRARY statement. They are best suited for analytic operations that are difficult to perform in SQL, or need to be performed frequently enough that their speed is a major concern.
Up-To-Date (Projection)

A projection is up-to-date (or up to date) if it is eligible to participate in query execution. Projections on empty tables are up-to-date upon creation. If the table has data loaded already, newly created projections are marked not up-to-date until refreshed. If a projection is refreshed while a node is down, that projection can be marked up-to-date even though it is missing data on one of the nodes. This is because the node will build the data during the recovery process before participating in queries.
**User-Defined SQL Function**

User-Defined SQL Functions let you define and store commonly-used SQL expressions as a function. User-Defined SQL Functions are useful for executing complex queries and combining Vertica built-in functions. You simply call the function name you assigned in your query.

A User-Defined SQL Function can be used anywhere in a query where an ordinary SQL expression can be used, except in the table partition clause or the projection segmentation clause.
View

A named logical relation specified by an associated query that can be accessed similarly to a table in the FROM clause of a SQL statement. The results of the query are not stored but obtained on the fly when the SQL referencing the view is executed.
Volatile functions

Regardless of their arguments or environment, volatile functions can return a different result with each invocation—for example, `UUID GENERATE()`.
vsql

vsql is a character-based, interactive, front-end utility that lets you type SQL statements and see the results. It also provides a number of meta-commands and various shell-like features that facilitate writing scripts and automating a variety of tasks.
Window (analytic)

An analytic function's OVER clause specifies how to partition, sort, and frame function input with respect to the current row. The input data is the result set that the query returns after it evaluates FROM, WHERE, GROUP BY, and HAVING clauses.
**Workload Analyzer (WLA)**

An advisor tool that analyzes system information held in SQL system tables (monitoring APIs) and returns a set of tuning recommendations. See Analyzing Workloads in the Administrator's Guide for details.
WOS (Write Optimized Store)

Write Optimized Store (WOS) is a memory-resident data structure for short-term data storage. The WOS is typically accessed by data manipulation language (DML) statements that load or remove data: INSERT, COPY, UPDATE, and DELETE. To support very fast data load speeds, WOS stores records without data compression or indexing. A projection in WOS is sorted only when it is queried. It remains sorted as long as no further data is inserted into it. WOS organizes data by epoch and holds both committed and uncommitted transaction data.

For more information, see Use of WOS for Trickle Loads.
Third-Party Software Acknowledgements

This document contains the licenses and notices for open source software used in Vertica Analytics Platform 9.0.x.

You can download open source code from https://my.vertica.com/licenses/.

[[[Undefined variable _Branding_Variables._Company_Legal_Name]]] makes no representations or warranties regarding any third party software. All third-party software is provided or recommended by Vertica on an AS IS basis.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).
Requesting Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by [[[Undefined variable _Branding_Variables._Company_Legal_Name]]].

To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett Packard Enterprise
Attn: General Counsel
3000 Hanover Street
Palo Alto, CA 94304
USA

Please specify the product and version for which you are requesting source code. You may also request a copy of this source code free of charge at: http://hpe.com/software/opensource.
Angularjs

A Webjar for AngularJS.

MIT License (MIT)Copyright (c) 2017Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Angular-bootstrap


MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Angular-multi-step-form

Angular module for creating wizards and multi-step forms, and licensed under ISC License (ISC):

Copyright © 2004-2013 by Internet Systems Consortium, Inc. (“ISC”)

Copyright © 1995-2003 by Internet Software Consortium

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Angular-ui-grid


MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Ansible

Copyright (c) 2017 Ansible Project.

Description

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and
control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any additional terms added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.
"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.
If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License. An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give
under the previous paragraph, plus a right to possession of the Corresponding Source of the
work from the predecessor in interest, if the predecessor has it or can get it with reasonable
efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed
under this License. For example, you may not impose a license fee, royalty, or other charge for
exercise of rights granted under this License, and you may not initiate litigation (including a
cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making,
using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a
work on which the Program is based. The work thus licensed is called the contributor's
"contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the
contributor, whether already acquired or hereafter acquired, that would be infringed by some
manner, permitted by this License, of making, using, or selling its contributor version, but do
not include claims that would be infringed only as a consequence of further modification of the
contributor version. For purposes of this definition, "control" includes the right to grant patent
sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the
contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise
run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment,
however denominated, not to enforce a patent (such as an express permission to practice a
patent or covenant not to sue for patent infringement). To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a patent against the
party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding
Source of the work is not available for anyone to copy, free of charge and under the terms of
this License, through a publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to
deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a
manner consistent with the requirements of this License, to extend the patent license to
downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the
patent license, your conveying the covered work in a country, or your recipient’s use of the
covered work in a country, would infringe one or more identifiable patents in that country that
you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or
propagate by procuring conveyance of, a covered work, and grant a patent license to some of
the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of
any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS
Apache-Arrow

Version 0.2.0
Apache License 2.0

Apache-arrow 0.2.0 is an early release and the APIs are still evolving.

Description

http://www.apache.org/licenses/

Version 2.0, January 2004
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or
other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work 
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or 
malfunction, or any and all other commercial damages or losses), even if such Contributor has 
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative 
Works thereof, You may choose to offer, and charge a fee for, acceptance of support, 
warranty, indemnity, or other liability obligations and/or rights consistent with this License. 
However, in accepting such obligations, You may act only on Your own behalf and on Your sole 
responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, 
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted 
against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Avalon Framework

Copyright © 1999-2004, Apache Software Foundation.

All rights reserved.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Avro

Version 1.7.0

Copyright © 2013 The Apache Software Foundation.

Apache Hadoop, Hadoop, HDFS, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Zookeeper are trademarks of the Apache Software Foundation.

http://avro.apache.org/

Description

Data serialization system: Apache Avro is a data serialization system. Avro provides:

- Rich data structures.
- A compact, fast, binary data format.
- A container file, to store persistent data.
- Remote procedure call (RPC).
- Simple integration with dynamic languages.

Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

Declared Fedora Licences: ASL 2.0

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Commons Collections

Copyright © 2001-2008 The Apache Software Foundation

The Java Collections Framework was a major addition in JDK 1.2. It added many powerful data structures that accelerate development of most significant Java applications. Since that time it has become the recognized standard for collection handling in Java. Commons-Collections seek to build upon the JDK classes by providing new interfaces, implementations and utilities. For more information, see https://commons.apache.org/proper/commons-collections/.

License Acknowledgments

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Commons Configuration

Copyright © 2001-2012 The Apache Software Foundation. All Rights Reserved

Version 1.6

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTicular PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Commons DBCP

Copyright © 2001-2014 The Apache Software Foundation. All Rights Reserved.

Version 1.4
Version 2.0.1

DBCP 2 binaries should be used by applications running under Java 7.
DBCP 1.4 binaries should be used by applications running under Java 6.

License Acknowledgments

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Commons FileUpload

Copyright © 2002-2010 The Apache Software Foundation. All Rights Reserved.

Version 1.2

The Commons FileUpload package makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.

License Acknowledgments

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Commons Logging

Copyright © 2001-2008 The Apache Software Foundation

The Apache Commons Logging component is a thin adapter allowing configurable bridging to other, well-known logging systems.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Derby

Copyright © 2004-2012 Apache Software Foundation

Version 10.6.2.1

Apache Derby, an Apache DB subproject, is an open source relational database, implemented entirely in Java. Vertica uses Apache Derby in Management Console (MC).

License Acknowledgments

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Hadoop

Copyright © 2012 The Apache Software Foundation.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems, and issue tracking
systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and
improving the Work, but excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a
Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display,
publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or
Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable (except as stated in this section) patent license to make, have made, use, offer
to sell, sell, import, and otherwise transfer the Work, where such license applies only to those
patent claims licensable by such Contributor that are necessarily infringed by their Contribution
(s) alone or by combination of their Contribution(s) with the Work to which such Contribution
(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or
counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the
Work constitutes direct or contributory patent infringement, then any patent licenses granted
to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works
thereof in any medium, with or without modifications, and in Source or Object form, provided
that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the
files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright,
patent, trademark, and attribution notices from the Source form of the Work, excluding those
notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works
that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative
Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Hadoop Libhdfs++

Apache License 2.0, Apache component version 2.7.2.

Vertica provides the libhdfs++ library to support C++ calls to the Apache Hadoop Distributed File System (HDFS). This version has new functionality and substantial modifications.

Description

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or
other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Hadoop Libhdfs++ Kerberos

Apache License 2.0, Apache component version 2.7.2

Vertica provides Kerberos support for the libhdfs++ library to support C++ calls to the Apache Hadoop Distributed File System (HDFS). This version has new functionality and substantial modifications.

Description

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache Hive

Copyright © 2013 The Apache Software Foundation.

Apache Hadoop, Hadoop, HDFS, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Zookeeper are trademarks of the Apache Software Foundation.

http://hive.apache.org/
http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Apache ORC

Apache License 2.0, Copyright © 2004
All rights reserved.

Description

The smallest, fastest columnar storage for Hadoop workloads.

Apache License Version 2.0

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.
"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You do not modify the use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NONINFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache ORC Subcomponents

The Apache ORC project contains subcomponents with separate copyright notices and license terms. Your use of the source code for the these subcomponents is subject to the terms and conditions of the following licenses.

Parts of the site formatting are licensed under the MIT License (MIT):

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Apache Parquet

Apache License 2.0, Parquet Component 2.3

Apache Parquet is a general-purpose columnar storage format, built for Hadoop. You can use Parquet with any data processing framework, data model, or programming language. The Apache Parquet library lets Vertica users read Parquet files efficiently. This work includes building an optimized reader in C++ for parquet files.

Description

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
ASMJIT

Copyright © 2008-2010, Petr Kobalicek <kobalicek.petr@gmail.com>
All rights reserved.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Copyright © 2000-2011 INRIA, France Telecom
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
asm.asm-commons

Copyright © 2000-2011 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
asm.asm-tree

Copyright © 2000-2011 INRIA, France Telecom

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
reward, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
**AWS-AWS-sdk-cpp**

Version 1.0.34

Apache License 2.0


**Description**

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or
other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and
   You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
BLAS

Version 3.6.0, updated November 2015

Basic Linear Algebra Subprograms (BLAS) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

License and Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. ASC-9313958 and DOE Grant No. DE-FG03-94ER25219. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF) or the Department of Energy (DOE).

The reference BLAS is a freely-available software package. It is available from netlib via anonymous ftp and the World Wide Web. Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.

Like all software, it is copyrighted. It is not trademarked, but we do ask the following:

- If you modify the source for these routines we ask that you change the name of the routine and comment the changes made to the original.

- We will gladly answer any questions regarding the software. If a modification is done, however, it is the responsibility of the person who modified the routine to provide support.
Boost

Boost provides free peer-reviewed portable C++ source libraries. For more information, see www.boost.org.

Version 1.59 - August 13th, 2015
Version 1.38 - February 8th, 2009

Copyright Beman Dawes, David Abrahams, 1998-2005.
Copyright Rene Rivera 2004-2005.

Boost Software License

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Bootstrap

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile, first projects on the web.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Boto

Copyright © 2009, 2010, Mitch Garnaat.
All rights reserved.
Version 2.48.0

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Bottle

Copyright © 2011, Pierre Quentel (pierre.quentel@gmail.com)

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**Brotil**

Brotil is a Google data encoding and compression format, used primarily to improve performance in modern browsers.

**MIT License (MIT)**

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
bzip2

Copyright © 1996-2005 Julian R Seward

This file is a part of bzip2 and/or libbzip2, a program and library for lossless, block-sorting data compression.

Copyright © 1996-2005 Julian R Seward. All rights reserved.

1. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

2. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

3. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

4. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

5. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, Cambridge, UK.

jseward@bzip.org <mailto:jseward@bzip.org>

bzip2/libbzip2 version 1.0 of 21 March 2000

This program is based on (at least) the work of:

Mike Burrows

David Wheeler
Peter Fenwick
Alistair Moffat
Radioed Neal
Ian H. Witten
Robert Sedgewick
Jon L. Bentley
cmaek

Cross Platform Makefile Generator

Copyright 2000-2009 Kitware, Inc., Insight Software Consortium

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
curl

Version 7.29.0

Copyright (c) 1996 - 2016, Daniel Stenberg, <daniel@haxx.se>.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.
curl_fopen.c

Copyright (c) 2003 Simtec Electronics

Re-implemented by Vincent Sanders<vince@kyllikki.org> with extensive reference to original curl example code

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR `"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
cyrus-sasl-library

Version: 2.1.26

For Vertica SQL on Hadoop, this is the Cyrus SASL API implementation. Use this library on the client or server side to provide authentication. This software package contains encryption software. Be sure to abide by appropriate export rules if you download it.

For more information, see http://asg.web.cmu.edu/sasl/sasl-library.html.

Description

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
d3js

Copyright 2010 - 2016 Michael Bostock
Version 3

License Acknowledgments

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Daemonize

Copyright © 2003-2011
Brian M. Clapper. All rights reserved

This software runs a command as a Unix daemon.

License Acknowledgments

With the exception of the install-sh script and the getopt.c source, this software is released under BSD license, which follows:

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Derby

Copyright © Apache Software Foundation
Version 10.6.1.0
http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of

Vertica Documentation
Third-Party Software Acknowledgements
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customey use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Flex Extractor Functions

Libraries to support extracting flex Vmaps include:

- MAPJSONEXTRACTOR
- MAPDELIMITEDEXTRACTOR
- MAPREGEXEXTRACTOR

License

http://www.apache.org/licenses/
Version 2.0, January 2004
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Flex Parsers

Parsers for unstructured data for flex and columnar tables are as follows:

- Avro
- CEF
- CSV
- Delimited
- JSON
- RegEx

License

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANDIBILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Flex Vmap Functions

Libraries to perform functions on Vmaps:

- EMPTYMAP
- MAPAGGREGATE
- MAPCONTAINSKEY
- MAPCONTAINSVALUE
- MAPITEMS
- MAPKEYS
- MAPKEYSINFO
- MAPLOOKUP
- MAPSIZE
- MAPTOSTRING
- MAPVALUES
- MAPVERSION

License

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was/were submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and

   You must cause any modified files to carry prominent notices stating that You changed the files; and

   You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Flex Vmaps

Table column format able to hold unstructured data. Vmap data does not require a schema definition.

License

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of
this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution (s) alone or by combination of their Contribution(s) with the Work to which such Contribution (s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You use, reproduce, and distribute the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate andgrossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Font Awesome

Copyright © 2014 Font Awesome by Dave Gandy - http://fontawesome.io

Version 4.3.0

MIT License (MIT)

Copyright (c) 2017 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
FreeMarker

Apache FreeMarker is a template engine: a Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data. For MC, FreeMarker is used to develop the threshold notification email template.

License Information

Copyright 2014 Attila Szegedi, Daniel Dekany, Jonathan Revusky
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.
   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.
   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.
   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.
"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You do not modify the License. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

=============================================================================
=  

FreeMarker Subcomponents with Different Copyright Owners

FreeMarker, both in its source code and binary form (freemarker.jar) includes a number of files that are licensed by the Apache Software Foundation under the Apache License, Version 2.0. This is the same license as the license of FreeMaker, but the copyright owner is the Apache Software Foundation. These files are:

- freemarker/ext/jsp/web-app_2_2.dtd
- freemarker/ext/jsp/web-app_2_3.dtd
- freemarker/ext/jsp/web-jsptaglibrary_1_1.dtd
- freemarker/ext/jsp/web-jsptaglibrary_1_2.dtd

Historical Notes

-------------
FreeMarker 1.x was released under the LGPL license. Later, by community consensus, we have switched over to a BSD-style license. As of FreeMarker 2.2pre1, the original author, Benjamin Geer, has relinquished the copyright in behalf of Visigoth Software Society. With FreeMarker 2.3.21 the license has changed to Apache License, Version 2.0, and the owner has changed from Visigoth Software Society to three of the FreeMarker 2.x developers, Attila Szegedi, Daniel Dekany, and Jonathan Revusky.
Ganglia Open Source License

Copyright © 2001 by Matt Massie and The Regents of the University of California.

All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without written agreement is hereby granted, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
GEOS

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the GEOS package can be found at http://my.vertica.com/licenses/geos-3.3.8.tar.bz2.
gperftools-libs

Version: 2.1

(Originally, Google Performance Tools), gperftools is a collection of a high-performance multi-threaded malloc() implementation, described as a thread-friendly heap-checker, heap-profile, and cpu-profiler. Includes tcmalloc, libtcmalloc, and libprofiler.

Description

Copyright (c) 2005, Google Inc.
All rights reserved.
BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Gradle-Node-Plugin

This is the Gradle plugin for running NodeJS scripts.

Copyright © Apache Software Foundation

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and

   You must cause any modified files to carry prominent notices stating that You changed the files; and

   You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Guava Libraries

Version 18.0

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
HCatalog

Version 0.5.0

Copyright © 2012-2013 The Apache Software Foundation

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Hibernate Validator

Copyright © 2009 - 2013 Red Hat, Inc. & Gunnar Morling

Version 5.1.3

http://www.apache.org/licenses/Version 2.0, January 2004 TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION 1. Definitions. "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License. "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License. "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files. "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types. "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below). "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof. "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution." "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work. 2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a
perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form. 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed. 4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions: You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License. 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions. 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as
required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.END OF TERMS AND CONDITIONS
html-minifier

Version 3.3.1


MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software
and associated documentation files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


**HttpClient**

Copyright Apache Software Foundation

Version 4.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and

   You must cause any modified files to carry prominent notices stating that You changed the files; and

   You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
wbarranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Httpcore

Copyright Apache Software Foundation

Version 4.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuous or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
httplib2

Copyright © 2006, Joe Gregorio (joe@bitworking.org)

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by its Contribution(s) alone or by combination of its Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
ICU (International Components for Unicode) License - ICU 1.8.1 and Later

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2009 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.
JSON.simple

Copyright © 2009, Yidong Fang, Chris Kokleberg
A simple Java toolkit for JSON.

License Acknowledgments

http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of
this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
javax.el.el-api

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. “Contributor” means each individual or entity that creates or contributes to the creation of Modifications.

1.2. “Contributor Version” means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. “Covered Software” means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. “Executable” means the Covered Software in any form other than Source Code.

1.5. “Initial Developer” means the individual or entity that first makes Original Software available under this License.

1.6. “Larger Work” means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. “License” means this document.

1.8. “Licensable” means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. “Modifications” means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. “Original Software” means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. “Patent Claims” means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. “Source Code” means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. “You” (or “Your”) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this
definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).
(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients’ rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.
You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient’s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.


You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3 Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.
COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABILITY, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as “Participant”) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY’S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS
SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” (as that term is defined at 48 C.F.R. § 252.227-7014(a)(1)) and “commercial computer software documentation” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction’s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys’ fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to
the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification"). Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections
when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However,
parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those
countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting: http://my.vertica.com/licenses/el-api-1.1.jar
javax.inject

Copyright 2001-2012 Bob Lee

Version 1

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
waranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
javax.transaction.jta

Copyright © 2002 Sun Microsystems, Inc. All rights reserved.

Java(TM) Transaction API (JTA) Specification ('Specification')
Version: 1.0.1B
Status: Maintenance Release
Release: November 5, 2002
Copyright 2002 Sun Microsystems, Inc.
4150 Network Circle, Santa Clara, California
95054, U.S.A
All rights reserved.

NOTICE; LIMITED LICENSE GRANTS

Sun Microsystems, Inc. ('Sun') hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under the Sun's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation, which shall be understood to include developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

Sun also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or patent rights it may have in the Specification to create and/or distribute an Independent Implementation of the Specification that: (i) fully implements the Spec(s) including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (iii) passes the TCK (including satisfying the requirements of the applicable TCK Users Guide) for such Specification. The foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose.

You need not include limitations (i)-(iii) from the previous paragraph or any other particular 'pass through' requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to implementations of the Specification (and products derived from them) that satisfy limitations (i)-(iii) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Sun's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Spec in question.
For the purposes of this Agreement: 'Independent Implementation' shall mean an implementation of the Specification that neither derives from any of Sun's source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun's source code or binary code materials; and 'Licensor Name Space' shall mean the public class or interface declarations whose names begin with 'java', 'javax', 'com.sun' or their equivalents in any subsequent naming convention adopted by Sun through the Java Community Process, or any recognized successors or replacements thereof.

This Agreement will terminate immediately without notice from Sun if you fail to comply with any material provision of or act outside the scope of the licenses granted above.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED 'AS IS'. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later
versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your use of the Specification ('Feedback'). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.
Jdom

Version 1.1.3

Copyright © 2011 Jason Hunter, Brett McLaughlin. All Rights Reserved.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
 WARRANTY, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
jemalloc

Version 4.2.1

This software is the jemalloc at github.com. See http://www.canonware.com/jemalloc/

Open Source Component jemalloc

Unless otherwise specified, files in the jemalloc source distribution are subject to the following license:

Copyright (C) 2002-2016 Jason Evans <jasone@canonware.com>. All rights reserved.

Copyright (C) 2007-2012 Mozilla Foundation. All rights reserved.

Copyright (C) 2009-2016 Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
BSD 2-clause "Simplified" License

All rights reserved.

See http://spdx.org/licenses/BSD-2-Clause

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Jetty

Copyright ©, Eclipse Foundation

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the
Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.
Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under
this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
jQuery

Copyright © 2011 jQuery Team
All rights reserved.
Version 2.1.0

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
jQuery-Selectmenu

Copyright © 2011, Paul Bakaus, http://jqueryui.com/

This software consists of voluntary contributions made by many individuals (AUTHORS.txt, http://jqueryui.com/about) For exact contribution history, see the revision history and logs, available at http://jquery-ui.googlecode.com/svn/

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Jquery-Sparklines

Copyright © 2011, jQuery Team

BSD 2-clause "Simplified" License

All rights reserved.

See http://spdx.org/licenses/BS2-Clause

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
jQuery Steps

Use the jQuery Wizard widget for MC Data Ingest tool.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**JUnitJS**

Copyright © 2011 JS Foundation

https://js.foundation/

Version 1.20.0


**MIT License (MIT)**

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
json-cpp

Version 0.6.0-rc2
Copyright (c) 2007-2010 Baptiste Lepilleur

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
json-org-java

Version 20140107

Copyright (c) 2002 JSON.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
junitjs

Copyright © 2011 JS Foundation
https://js.foundation/
Version 1.20.0


MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Copyright (C) 2000-2007 the JWNL development team
(http://www.sourceforge.net/projects/jwordnet)
All rights reserved.
Version 1.3.3
BSD 3-clause "New" or "Revised" License
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Karma

Spectacular Test Runner for JavaScript

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Chrome-Launcher

A Karma plugin. Launcher for Chrome and Chrome Canary.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Coverage

A Karma plugin to generate code coverage.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Firefox-Launcher

A Karma plugin launcher for Firefox.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Jasmine

A Karma plugin - adapter for Jasmine testing framework.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Junit-Reporter

A Karma plugin to report results in junit xml format.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-ng-html2js-preprocessor

A Karma plugin. Adapter for QUnit testing framework.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-Phantomjs-Launcher

A Karma plugin. Launcher for PhantomJS.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Karma-qunit

A Karma plugin. Adapter for QUnit testing framework.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Copyright (c) 2010-2011 Cybozu Labs, Inc. All rights reserved.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
LAPACK

Version: 3.4.0
Updated November 11, 2011

LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision. Vertica statically links this to other code (compiled code co-mingled in the same binary files).

Description

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**Libcsv**

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the libcsv dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://my.vertica.com/licenses/libcsv-3.0.1.tar.gz
libgfortran

Open source component version 4.8.2

Description

The libgfortran contains a Fortran shared library, needed to run Fortran dynamically-linked programs, required for the Vertica LAPACK library addition.

The GNU General Public License (GPL) with Special Library Exception

The author has provided an exception to the GPL allowing the library to be linked with other programs provided that some specific conditions are met. Refer to the license text.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the
original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display
an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or
binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of
software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA
BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
Lighttpd Open Source License

Copyright © 2004, Jan Kneschke, incremental

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the 'incremental' nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
lib-javascript-jqplot

Copyright (c) 2009-2010 Chris Leonello
Version 1.0.8

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libio

Version 1.0
Copyright (c) 2013 Joos Kiener <Joos.Kiener@gmail.com>

Description

IO: Classe(s) for improving IO in Java. Currently there is only 1 class in this library.


This software consists of voluntary contributions made by many individuals (AUTHORS.txt, http://jqueryui.com/about) For exact contribution history, see the revision history and logs, available at http://jquery-ui.googlecode.com/svn/

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libjackson-java

Version 2.3

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the pyOpenSSL package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at these locations:

https://my.vertica.com/licenses/jackson-core-asl-1.8.8.jar

https://my.vertica.com/licenses/jackson-mapper-asl-1.8.8.jar

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANDABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
libjs-codemirror

Copyright (C) 2014 by Marijn Haverbeke <marijnh@gmail.com> and others
Version 2.0

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**libjs-jquery-datatables.columnfilters**

Version 1.5.5

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
libjs-infovis-toolkit

Copyright © 2013 SenchaLabs - Author: Nicolas Garcia Belmonte

Version 2.0.1

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libjs-jquery-slider

Copyright © 2012 Egor Khmelev
Version 1.1.0

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libjs-jquery-tiptip

Copyright © 2010 Drew Wilson.
Version 1.3

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.
To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.


The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.
The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.
You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the
software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term
(regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing
other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.
If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Afferro General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE StATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS
libjs-jquery-ui
Version 1.10.4

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libjs-jquery-ui-multiselect

Copyright (c) 2011 Eric Hynds
Version 1.1.4

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Libopencsv-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved.
Version 2.3

Description

opencsv - Library for reading and writing CSV in Java

Opencsv is a very simple csv (comma-separated values) parser library for Java. It supports all the basic csv-type things you're likely to want to do:

- Arbitrary numbers of values per line
- Ignoring commas in quoted elements
- Handling quoted entries with embedded carriage returns (ie entries that span multiple lines)
- Configurable separator and quote characters (or use sensible defaults)
- Read all the entries at once, or use an Iterator style model
- Creating csv files from String[] (ie. automatic escaping of embeddedquote chars)

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.
"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution
(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
**librdkafka-dev**

Version 0.8.6
Copyright (c) 2015

**Description**

Library implementing the Apache Kafka protocol (development headers), librdkafka is a C implementation of the Apache Kafka protocol. It currently implements the 0.8 version of the protocol and can be used to develop both Producers and Consumers.

More information about Apache Kafka can be found at http://kafka.apache.org/

This package contains the development headers.

Declared Ubuntu Licences: BSD-2-clause, BSD-3-clause, MIT

**MIT License (MIT)**

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**libsnpappy1**

Version 1.1.2  
Copyright (c) 2015

**Description**

Snappy (libsnpappy1) is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. For instance, compared to the fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

Snappy is widely used inside Google, in everything from BigTable and MapReduce to our internal RPC systems. (Snappy has previously been referred to as “Zippy” in some presentations and the likes.)

Copyright 2011, Google Inc.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
libtar

Version 1.2.20
Copyright (c) 2015

Description

Tar file manipulation API libtar is a C library for manipulating tar archives. It supports both the strict POSIX tar format and many of the commonly-used GNU extensions.

This software consists of voluntary contributions made by many individuals (AUTHORS.txt, http://jqueryui.com/about) For exact contribution history, see the revision history and logs, available at http://jquery-ui.googlecode.com/svn/

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
libtcmalloc-minimal4

Version 2.0

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Libthttpclient-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved

Version 4.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   You must give any other recipients of the Work or Derivative Works a copy of this License; and
   You must cause any modified files to carry prominent notices stating that You changed the files; and
   You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warrenty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Libthttpcore-Java

Copyright © 2005-2012 The Apache Software Foundation. All Rights Reserved

Version 4.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Libhttpmime-Java

Copyright © 1999-2012 The Apache Software Foundation. All Rights Reserved.

Version 4.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
libuv

Copyright 2015 Joyent, Inc, Node.js is a trademark of Joyent, Inc.
Version 1.0.0

Description

Platform layer for node.js.

libuv is a new platform layer for Node. Its purpose is to abstract IOCP on Windows and libev on Unix systems. We intend to eventually contain all platform differences in this library.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
log4j.log4j

Copyright © 1999-2010 Apache Software Foundation

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems, and issue tracking
systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and
improving the Work, but excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a
Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display,
publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or
Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable (except as stated in this section) patent license to make, have made, use, offer
to sell, sell, import, and otherwise transfer the Work, where such license applies only to those
patent claims licensable by such Contributor that are necessarily infringed by their Contribution
(s) alone or by combination of their Contribution(s) with the Work to which such Contribution
(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or
counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the
Work constitutes direct or contributory patent infringement, then any patent licenses granted
to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works
thereof in any medium, with or without modifications, and in Source or Object form, provided
that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the
files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright,
patent, trademark, and attribution notices from the Source form of the Work, excluding those
notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works
that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative
Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
logkit.logkit

Copyright © 2004 Apache Software Foundation.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Lpsolve

Package: lpSolve
Version: 5.6.6
Date: 2011-04-19
Title: Interface to lp_solve v. 5.5 to solve linear/integer programs
Author: Michel Berkelaar and others
Maintainer: Sam Buttrey <buttrey@nps.edu>
version 5.5.

Description

License: LGPL-2
Date/Publication: 2011-04-26 06:30:54

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Lpsolve package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://my.vertica.com/licenses/lpSolve_5.6.6.tar.gz.
**Lpsolveapi**

Package: lpSolveAPI
Version: 5.5.2.0-5
Date: 2011-07-28
Title: R Interface for lp_solve version 5.5.2.0
Author: lp_solve <http://lp.solve.sourceforge.net/>, Kjell Konis <kjell.konis@epfl.ch>.
Maintainer: Kjell Konis <kjell.konis@epfl.ch>
License: LGPL-2
Date/Publication: 2011-08-03 11:41:56
Packaged: 2011-07-28 21:09:07 UTC; rforge

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Lpsolveapi package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://my.vertica.com/licenses/lpSolveAPI_5.5.2.0-5.tar.gz.
Math-Atlas

Open source component version: 3.11.30

Vertica LAPACK library addition. ATLAS ( Automatically Tuned Linear Algebra Software) provides optimized Linear Algebra kernels for arbitrary cache-based architectures. ATLAS provides ANSI C and Fortran77 interfaces for the entire BLAS API, and a small portion of the LAPACK AP.

Description

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
The Apache Software Foundation.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
MersenneTwister.h

Copyright © 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
Copyright © 2000 - 2009, Richard J. Wagner

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
MIT Kerberos

Version 1.11.3

Copyright © 1985-2013 by the Massachusetts Institute of Technology.

Export of software employing encryption from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original MIT software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Individual source code files are copyright MIT, Cygnus Support, Novell, OpenVision Technologies, Oracle, Red Hat, Sun Microsystems, FundsXpress, and others.

Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

“Commercial use” means use of a name in a product or other for-profit manner. It does NOT prevent a commercial firm from referring to the MIT trademarks in order to convey information (although in doing so, recognition of their trademark status should be given).
Mockito

Mocking framework for unit tests in Java.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**Msgpack**

Msgpack is used in Vertica Side Process.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
net.sf.json

Copyright © 2006-2010 Andres Almmiray

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Npm-html2js

Version 0.1.8

Standalone script to load Angular html/jade templates:
https://www.npmjs.com/package/npm-html2js

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Oauth

Copyright © 2007, Leah Culver

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
oauth2

Copyright © 2011, Joe Stump

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Open LDAP

Copyright © The OpenLDAP Public License

Version 2.8, 17 August 2003

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

Redistributions in source form must retain copyright statements and notices

Redistributions in binary form must reproduce applicable copyright statements and notices, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution, and

Redistributions must contain a verbatim copy of this document.

The OpenLDAP Foundation may revise this license from time to time. Each revision is distinguished by a version number. You may use this Software under terms of this license revision or under the terms of any subsequent revision of the license.

THIS SOFTWARE IS PROVIDED BY THE OPENLDAP FOUNDATION AND ITS CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENLDAP FOUNDATION, ITS CONTRIBUTORS, OR THE AUTHOR(S) OR OWNER(S) OF THE SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The names of the authors and copyright holders must not be used in advertising or otherwise to promote the sale, use or other dealing in this Software without specific, written prior permission. Title to copyright in this Software shall at all times remain with copyright holders.

OpenLDAP is a registered trademark of the OpenLDAP Foundation.

Copyright 1999-2003 The OpenLDAP Foundation, Redwood City, California, USA. All Rights Reserved. Permission to copy and distribute verbatim copies of this document is granted.
openlp

Copyright © 2012 The Apache Software Foundation.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
OpenSSL License

Version 1.0.2b

Copyright © 1998-2016 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)

The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
org.apache.geronimo.specs.geronimo-jta_1.1_spec

Copyright © 2003-2011, The Apache Software Foundation

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NONINFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
org.apache.ibatis.ibatis-sqlmap

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole
responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the
Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.
Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under
this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:
   i) changes to the Program, and
   ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor’s behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the
Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.
Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient’s patent(s), then such Recipient’s rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under
this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
org.codehaus.jackson.jackson-core-asl

Copyright ©2009-2011, FasterXML, LLC

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by its Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
org.codehaus.jackson.jackson-mapper-asl

Copyright ©2009-2011 FasterXML, LLC
http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersedes or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
org.eclipse.jdt.core.compiler.ecj

Copyright © 2011 The Eclipse Foundation. All Rights Reserved.

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the
Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.
Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under
this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
org.glassfish.web.jstl-impl

Copyright © 2008, 2010 Oracle and/or its affiliates. All rights reserved. Use is subject to license terms.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
org.quartz-scheduler.quartz

Copyright © 2011, Terracotta, Inc.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
orion-ssh2

Copyright © 2010, Juraj Bednar, Trilead AG

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**osgeo-proj.4**

PROJ.4 - Cartographic Projections Library, as MIT License (also X11). Vertica uses open-source libraries for geospatial functions.

**MIT License (MIT)**

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Paramiko

Copyright © 20011, Robey Pointer

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the paramiko dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://www.my.vertica.com/licenses/paramiko-1.7.7.1.tar.gz
Copyright ©, Ian Bicking

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
PCRE LICENCE

Copyright © 1997-2010 University of Cambridge
Copyright © 2007-2010, Google Inc.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Perl Artistic License

Copyright © August 15, 1997

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
4. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

   a. use the modified Package only within your corporation or organization.

   b. rename any nonstandard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each nonstandard executable that clearly documents how it differs from the Standard Version.

   c. make other distribution arrangements with the Copyright Holder.

5. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

   a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

   b. accompany the distribution with the machine-readable source of the Package with your modifications.

   c. give nonstandard executables nonstandard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

   d. make other distribution arrangements with the Copyright Holder.

6. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

7. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-
called "undump" or "unexec" methods of producing a binary executable image, then
distribution of such an image shall neither be construed as a distribution of this Package
nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not
represent such an executable image as a Standard Version of this Package.

8. C subroutines (or comparably compiled subroutines in other languages) supplied by you
and linked into this Package in order to emulate subroutines and variables of the language
defined by this Package shall not be considered part of this Package, but are the equivalent
of input as in Paragraph 6, provided these subroutines do not change the language in any
way that would cause it to fail the regression tests for the language.

9. Aggregation of this Package with a commercial distribution is always permitted provided
that the use of this Package is embedded; that is, when no overt attempt is made to make
this Package's interfaces visible to the end user of the commercial distribution. Such use
shall not be construed as a distribution of this Package.

10. The name of the Copyright Holder may not be used to endorse or promote products
derived from this software without specific prior written permission.

**THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE.**

The End
Pexpect

Copyright © 2010 Noah Spurrier

Credits: Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, Fernando Perez, Corey Minyard, Jon Cohen, Guillaume Chazarain, Andrew Ryan, Nick Craig-Wood, Andrew Stone, Jorgen Grahn (Let me know if I forgot anyone.)

Free, open source, and all that good stuff.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Phantomjs

PhantomJS (phantomjs.org) is a headless WebKit, capable of scripting with JavaScript.

LICENSE.BSD

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
PHP License

Copyright © The PHP License, version 3.01

Copyright © 1999 - 2009 The PHP Group. All rights reserved.

1. Redistribution and use in source and binary forms, with or without modification, is permitted provided that the following conditions are met:

2. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

3. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

4. The name "PHP" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact group@php.net.

5. Products derived from this software may not be called "PHP", nor may "PHP" appear in their name, without prior written permission from group@php.net. You may indicate that your software works in conjunction with PHP by saying "Foo for PHP" instead of calling it "PHP Foo" or "phpfoo"

6. The PHP Group may publish revised and/or new versions of the license from time to time. Each version will be given a distinguishing version number.

Once covered code has been published under a particular version of the license, you may always continue to use it under the terms of that version. You may also choose to use such covered code under the terms of any subsequent version of the license published by the PHP Group. No one other than the PHP Group has the right to modify the terms applicable to covered code created under this License.

Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes PHP software, freely available from <http://www.php.net/software/>".

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PHP DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the PHP Group.

The PHP Group can be contacted via Email at group@php.net.

For more information on the PHP Group and the PHP project, please see <http://www.php.net>.

**PostgreSQL**

This product uses the PostgreSQL Database Management System (formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2005, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
Protobuf

Copyright 2008 Google Inc. All rights reserved.

Version 2.4.1, 2.6.0

Description

Protocol Buffers - Google's data interchange format.
 protocol buffers are a way of encoding structured data in an efficient yet extensible format. Google uses Protocol Buffers for almost all of its internal RPC protocols and file formats. Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data—think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages. You can even update your data structure without breaking deployed programs that are compiled against the "old" format.

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
protobuf-java

Copyright 2008 Google Inc. All rights reserved.

Version 2.4.1

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Pybonjour

Copyright ©, cstawarz@gmail.com

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Pyopenssl

Copyright ©, Jean-Paul Calderone

pyOpenSSL 0.11 is distributed under the GNU Lesser General Public License, below.

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the pyOpenSSL package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://www.my.vertica.com/licenses/pyOpenSSL-0.11.tar.gz.
PySNMP

Copyright © 1999-2012 Ilya Etingof

BSD 3-clause "New" or "Revised" License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Python

Copyright © 2001-2012 Python Software Foundation; All Rights Reserved
This is the official license for the Python 2.7.* release.

Description

Interactive high-level object-oriented language (default version).

Python, the high-level, interactive object oriented language, includes an extensive class library with lots of goodies for network programming, system administration, sounds and graphics.

This package is a dependency package, which depends on Debian's default Python version (currently v2.7).

History of the Software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python’s principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see http://www.zope.com/). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org/ for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

<table>
<thead>
<tr>
<th>Release</th>
<th>Derived from</th>
<th>Year</th>
<th>Owner</th>
<th>GPL-compatible? (1)</th>
</tr>
</thead>
</table>

Vertica Analytic Database (9.0.x)  Page 6042 of 6180
Note: 1. GPL-compatible doesn’t mean that we’re distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don’t.

2. According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido’s direction to make these releases possible.

**Terms and Conditions for Accessing or Otherwise Using Python**

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSFhereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.
BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com (“BeOpen”), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization (“Licensee”) accessing and otherwise using this software in source or binary form and its associated documentation (“the Software”).

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python”
logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI’s License Agreement and CNRI’s notice of copyright, i.e., “Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved” are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI’s License Agreement, Licensee may substitute the following text (omitting the quotes): “Python 1.6.1 is made available subject to the terms and conditions in CNRI’s License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013.”

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an “AS IS” basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia’s conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the “ACCEPT” button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.
STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
A. HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see http://www.zope.com). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

<table>
<thead>
<tr>
<th>Release</th>
<th>Derived From</th>
<th>Year</th>
<th>Owner</th>
<th>GPL-compatible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>thru 1.2</td>
<td>1991-1995</td>
<td>CWI</td>
<td>yes</td>
</tr>
<tr>
<td>1.3</td>
<td>thru 1.5.2</td>
<td>1.2</td>
<td>1995-1999</td>
<td>CNRI</td>
</tr>
<tr>
<td>1.6</td>
<td>1.5.2</td>
<td>2000</td>
<td>CNRI</td>
<td>no</td>
</tr>
<tr>
<td>2.0</td>
<td>1.6</td>
<td>2000</td>
<td>BeOpen.com</td>
<td>no</td>
</tr>
<tr>
<td>1.6.1</td>
<td>1.6</td>
<td>2001</td>
<td>CNRI</td>
<td>yes (2)</td>
</tr>
<tr>
<td>2.1</td>
<td>2.0+1.6.1</td>
<td>2001</td>
<td>PSF</td>
<td>no</td>
</tr>
<tr>
<td>2.0.1</td>
<td>2.0+1.6.1</td>
<td>2001</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.1</td>
<td>2.1+2.0.1</td>
<td>2001</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.2</td>
<td>2.1.1</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.3</td>
<td>2.1.2</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.2</td>
<td>and above</td>
<td>2.1.1</td>
<td>2001-now</td>
<td>PSF</td>
</tr>
</tbody>
</table>

Footnotes:
(1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with othersoftware that is released under the GPL; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON

================================================================================================

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

===============================================

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.


3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

------------------------------------------------------------------

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at
http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

---------------------------------------------

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

-----------------------------------------------

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Python Dialog

The Administration Tools part of this product uses Python Dialog, a Python module for doing console-mode user interaction.

Upstream Author:
Peter Astrand <peter@cendio.se>
Robb Shecter <robb@acm.org>
Sultanbek Tezadov
Florent Rougon <flo@via.ecp.fr>

Copyright © 2000 Robb Shecter, Sultanbek Tezadov

Copyright © 2002, 2003, 2004 Florent Rougon

License Description

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The complete source code of the Python dialog package and complete text of the GNU Lesser General Public License can be found on the Vertica Systems Web site at http://www.my.vertica.com/licenses/pythondialog-2.7.tar.bz2
Python Pip

Copyright © 2008-2016 The pip developers (see AUTHORS.txt).

Description

The Python pip module is for installing packages.

Installing the Vertica server RPM, includes the python distribution. Python is installed under the open source software (oss) directory, /opt/vertica/oss/python. A sub-directory of the python directory includes the pip module.

Copyright (c) 2008-2016 The pip developers (see AUTHORS.txt file)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

AUTHORS File Contents

For the most recent version of AUTHORS.txt, see https://github.com/pypa/pip/blob/develop/AUTHORS.txt.
Quartz
Copyright © 2001-2010 Terracotta, Inc.
Version 1.6.1

Description

http://www.apache.org/licenses/

Version 2.0, January 2004
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of
this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You use, reproduce, and distribute the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Quartz-scheduler-quartz

Code for Quartz scheduler
Version 1.8.5

Description

http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of
this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided You use, reproduce, and distribute the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
QUnit

Copyright 2013 The JQuery Foundation
All rights reserved.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
R

R version 3.0.0 (2013-04-03) -- "Masked Marvel"

Copyright (C) 2013 The R Foundation for Statistical Computing

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.


The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.
A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.
Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to
the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.
8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for
exercise of rights granted under this License, and you may not initiate litigation (including a

cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making,

using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a

work on which the Program is based. The work thus licensed is called the contributor's

"contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the

contributor, whether already acquired or hereafter acquired, that would be infringed by some

manner, permitted by this License, of making, using, or selling its contributor version, but do

not include claims that would be infringed only as a consequence of further modification of the

contributor version. For purposes of this definition, "control" includes the right to grant patent

sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the

contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise

run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment,

however denominated, not to enforce a patent (such as an express permission to practice a

patent or covenant not to sue for patent infringement). To "grant" such a patent license to a

party means to make such an agreement or commitment not to enforce a patent against the

party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding

Source of the work is not available for anyone to copy, free of charge and under the terms of

this License, through a publicly available network server or other readily accessible means,

then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to

deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a

manner consistent with the requirements of this License, to extend the patent license to
downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the

patent license, your conveying the covered work in a country, or your recipient's use of the

covered work in a country, would infringe one or more identifiable patents in that country that

you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or

propagate by procuring conveyance of, a covered work, and grant a patent license to some of

the parties receiving the covered work authorizing them to use, propagate, modify or convey a

specific copy of the covered work, then the patent license you grant is automatically extended
to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage,

prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights
that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

The complete source code of the R package can be found at https://my.vertica.com/licenses/R-2.14.0.tar.gz.

Sources for the Vertica User Defined Functions in R can be found at https://my.vertica.com/downloads/.
r-cran-rcpp

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers

Version 0.9.5

Description

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and
control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.


You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to
the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.
8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.


You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for
exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights
that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.


If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

You can download source code of the r-cran-rcpp package from: https://my.vertica.com/licenses/Rcpp_0.9.5.tar.gz.
Description

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Rinside

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers
Version 0.2.4

Description

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)
The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The complete source code of the rinside package can be found at
RRDTool Open Source License

Copyright © RRDTool Open Source License

Note: rrdtool is a dependency of using the ganglia-web third-party tool. RRDTool allows the graphs displayed by ganglia-web to be produced.

RRDTOOL - Round Robin Database Tool

A tool for fast logging of numerical data graphical display of this data.

Copyright © 1998-2008 Tobias Oetiker

All rights reserved.

Description

GNU GPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

FLOSS License Exception

(Adapted from http://www.mysql.com/company/legal/licensing/foss-exception.html)

I want specified Free/Libre and Open Source Software ("FLOSS") applications to be able to use specified GPL-licensed RRDtool libraries (the "Program") despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the "GPL").

As a special exception to the terms and conditions of version 2.0 of the GPL:

You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a "FLOSS Work") licensed under one or more of the licenses listed below, as long as:

1. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves

2. All identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves

   - are distributed subject to one of the FLOSS licenses listed below, and

   - the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections.

3. Any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL.

FLOSS License List

<table>
<thead>
<tr>
<th>License name</th>
<th>Version(s)/Copyright Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic Free License 2.0</td>
<td></td>
</tr>
<tr>
<td>Apache Software License 1.0/1.1/2.0</td>
<td></td>
</tr>
<tr>
<td>Apple Public Source License 2.0</td>
<td></td>
</tr>
<tr>
<td>Artistic license From Perl 5.8.0</td>
<td></td>
</tr>
<tr>
<td>BSD license &quot;July 22 1999&quot;</td>
<td></td>
</tr>
<tr>
<td>Common Public License 1.0</td>
<td></td>
</tr>
<tr>
<td>GNU Library or &quot;Lesser&quot; General Public License (LGPL) 2.0/2.1</td>
<td></td>
</tr>
<tr>
<td>IBM Public License, Version 1.0</td>
<td></td>
</tr>
<tr>
<td>Jabber Open Source License 1.0</td>
<td></td>
</tr>
<tr>
<td>MIT License (As listed in file MIT-License.txt) -</td>
<td></td>
</tr>
<tr>
<td>Mozilla Public License (MPL) 1.0/1.1</td>
<td></td>
</tr>
<tr>
<td>Open Software License 2.0</td>
<td></td>
</tr>
<tr>
<td>OpenSSL license (with original SSLeay license) &quot;2003&quot; (&quot;1998&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
PHP License 3.0
Python license (CNRI Python License) -
Python Software Foundation License 2.1.1
Sleepycat License "1999"
W3C License "2001"
X11 License "2001"
Zlib/libpng License -
Zope Public License 2.0/2.1
rsync

rsync was originally written by Andrew Tridgell and is currently maintained by Wayne Davison. It has been improved by many developers from around the world.

Description

rsync may be used, modified and redistributed only under the terms of the GNU General Public License, found at:

http://www.fsf.org/licenses/gpl.html

The GNU General Public License applies just to rsync and no other components of Vertica.

You are entitled to receive the source code for such software. For no less than three years from the date you obtained this software package, you may download a copy of the source code for the software in this package licensed under the GPL at no charge by visiting:

http://www.my.vertica.com/licenses/rsync-3.0.7.tar.gz
http://my.vertica.com/licenses/vertica_rsync_patch.txt

You can download this source code so it remains separate from other software on your computer system.
scons

Copyright (c) 2004-2011 All rights reserved
Version 2.2.0

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
selenium-htmlunit-driver

Copyright © 2012 The Apache Software Foundation.

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Setuptools

Copyright © 1990 - 2011, Philip J. Eby

Zope Public License (ZPL) Version 2.0

This software is Copyright (c) Zope Corporation (tm) and Contributors. All rights reserved.

This license has been certified as open source. It has also been designated as GPL compatible by the Free Software Foundation (FSF).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions in source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name Zope Corporation (tm) must not be used to endorse or promote products derived from this software without prior written permission from Zope Corporation.

4. The right to distribute this software or to use it for any purpose does not give you the right to use Servicemarks (sm) or Trademarks (tm) of Zope Corporation. Use of them is covered in a separate agreement (see http://www.zope.com/Marks).

5. If any files are modified, you must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

Disclaimer

THIS SOFTWARE IS PROVIDED BY ZOPE CORPORATION ``AS IS`` AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ZOPE CORPORATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of contributions made by Zope Corporation and many individuals on behalf of Zope Corporation. Specific attributions are listed in the accompanying credits file.
signpost-commonshttp4

Copyright 2012 Matthias Käppler

Version 1.2.1.1

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within
Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from
the Work, provided that such additional attribution notices cannot be construed as modifying
the License.

You may add Your own copyright statement to Your modifications and may provide additional
or different license terms and conditions for use, reproduction, or distribution of Your
modifications, or for any such Derivative Works as a whole, provided Your use, reproduction,
and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution
intentionally submitted for inclusion in the Work by You to the Licensor shall be under the
terms and conditions of this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify the terms of any
separate license agreement you may have executed with Licensor regarding such
Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks,
service marks, or product names of the Licensor, except as required for reasonable and
customary use in describing the origin of the Work and reproducing the content of the NOTICE
file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor
provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including,
without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,
MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for
determining the appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including
negligence), contract, or otherwise, unless required by applicable law (such as deliberate and
grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for
damages, including any direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to use the Work
(including but not limited to damages for loss of goodwill, work stoppage, computer failure or
malfunction, or any and all other commercial damages or losses), even if such Contributor has
been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative
Works thereof, You may choose to offer, and charge a fee for, acceptance of support,
warrenty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Signpost-Core

Copyright 2012 Matthias Käppler
Version 1.2.1.1
http://www.apache.org/licenses/
Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Signpost-Commonshttp 4 1.2.1.1

Copyright © 2012 Matthias Käppler

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
slf4j

Copyright (c) 2004-2013 QOS.ch
Version 1.7.5

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
slf4j-log4j12

Copyright (c) 2004-2013 QOS.ch

Version 1.7.5

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**slf4j-simple**

Copyright (c) 2004-2011 All rights reserved

Version 1.6.4

---

**MIT License (MIT)**

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Spring-Ldap

Copyright © 2005-2010 Mattias Arthursson, Ulrik Sandberg, Eric Dalquist, Keith Barlow

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Spring MVC

Copyright © 2011, Spring Source.org

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems, and issue tracking
systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and
improving the Work, but excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a
Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display,
publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or
Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable (except as stated in this section) patent license to make, have made, use, offer
to sell, sell, import, and otherwise transfer the Work, where such license applies only to those
patent claims licensable by such Contributor that are necessarily infringed by their Contribution
(s) alone or by combination of their Contribution(s) with the Work to which such Contribution
(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or
counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the
Work constitutes direct or contributory patent infringement, then any patent licenses granted
to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works
thereof in any medium, with or without modifications, and in Source or Object form, provided
that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the
files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright,
patent, trademark, and attribution notices from the Source form of the Work, excluding those
notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works
that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative
Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
springframework-security 3.2

Copyright © 2004-2010 Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Alef Arendsen, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abedrabbo, Andy Clement, Dave Syer, Oliver Gierke

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
Springframework-Tx 3.0.5

Copyright © 2004-2010 Rod Johnson, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Alef Arendsen, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma, Chris Beams, Tareq Abedrabbo, Andy Clement, Dave Syer, Oliver Gierke

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement You may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
springframework-webmvc

Version 4.0.0
Copyright © 2012
http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
spring-mobile

Copyright © 2012

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
**SNMP**

Various copyrights apply to this package, listed in various separate parts below. Please make sure that you read all the parts. Up until 2001, the project was based at UC Davis, and the first part covers all code written during this time. From 2001 onwards, the project has been based at SourceForge, and Networks Associates Technology, Inc hold the copyright on behalf of the wider Net-SNMP community, covering all derivative work done since then. An additional copyright section has been added as Part 3 below also under a BSD license for the work contributed by Cambridge Broadband Ltd. to the project since 2001. An additional copyright section has been added as Part 4 below also under a BSD license for the work contributed by Sun Microsystems, Inc. to the project since 2003.

Code has been contributed to this project by many people over the years it has been in development, and a full list of contributors can be found in the README file under the THANKS section.

Part 1: CMU/UCD copyright notice: (BSD like)


Copyright © 1996, 1998-2000 The Regents of the University of California

All Rights Reserved

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of CMU and The Regents of the University of California not be used in advertising or publicity pertaining to distribution of the software without specific written permission.

CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL CMU OR THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Part 2: Networks Associates Technology, Inc copyright notice (BSD)

Copyright © 2001-2003, Networks Associates Technology, Inc

All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the Networks Associates Technology, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 3: Cambridge Broadband Ltd. copyright notice (BSD)

Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The name of Cambridge Broadband Ltd. may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANDISABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 4: Sun Microsystems, Inc. copyright notice (BSD)

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Use is subject to license terms below.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 5: Sparta, Inc copyright notice (BSD)
Copyright © 2003-2006, Sparta, Inc
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Sparta, Inc nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 6: Cisco/BUPTNIC copyright notice (BSD)

Copyright © 2004, Cisco, Inc and Information Network Center of Beijing University of Posts and Telecommunications.
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Neither the name of Cisco, Inc, Beijing University of Posts and Telecommunications, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD)

Copyright © Fabasoft R&D Software GmbH & Co KG, 2003

oss@fabasoft.com

Author: Bernhard Penz

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries, brand or product names may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Standard

Copyright 2004-2012 Apache Software Foundation

Version 1.1.2

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is
intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.
However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
**Tecla Command-Line Editing**

Copyright © 2000 by Martin C. Shepherd.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.
twittersource.java

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of TITLE, NONINFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
twittersourceconstants.java

http://www.apache.org/licenses/
Version 2.0, January 2004
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.
"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.
"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.
"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems, and issue tracking
systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and
improving the Work, but excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a
Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display,
publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or
Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-
free, irrevocable (except as stated in this section) patent license to make, have made, use, offer
to sell, sell, import, and otherwise transfer the Work, where such license applies only to those
patent claims licensable by such Contributor that are necessarily infringed by their Contribution
(s) alone or by combination of their Contribution(s) with the Work to which such Contribution
(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or
counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the
Work constitutes direct or contributory patent infringement, then any patent licenses granted
to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works
thereof in any medium, with or without modifications, and in Source or Object form, provided
that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the
files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright,
patent, trademark, and attribution notices from the Source form of the Work, excluding those
notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works
that You distribute must include a readable copy of the attribution notices contained within
such NOTICE file, excluding those notices that do not pertain to any part of the Derivative
Works, in at least one of the following places: within a NOTICE text file distributed as part of
the Derivative Works; within the Source form or documentation, if provided along with the
Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational
purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties of OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
UI-Grid

MC uses UI-Grid v3.0.x for Data Ingest and Hadoop tables.

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
WebOb

Copyright © 2007 Ian Bicking, Sergey Schetinin

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Webware

Copyright © Ian Bicking and Contributors

History of the Software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python’s principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see http://www.zope.com/). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org/ for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

<table>
<thead>
<tr>
<th>Release</th>
<th>Derived from</th>
<th>Year</th>
<th>Owner</th>
<th>GPL-compatible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>thru 1.2</td>
<td>1991-1995</td>
<td>CWI</td>
<td>yes</td>
</tr>
<tr>
<td>1.3</td>
<td>thru 1.5.2</td>
<td>1995-1999</td>
<td>CNRI</td>
<td>yes</td>
</tr>
<tr>
<td>1.6</td>
<td>1.5.2</td>
<td>2000</td>
<td>CNRI</td>
<td>no</td>
</tr>
<tr>
<td>2.0</td>
<td>1.6</td>
<td>2000</td>
<td>BeOpen.com</td>
<td>no</td>
</tr>
<tr>
<td>1.6.1</td>
<td>1.6</td>
<td>2001</td>
<td>CNRI</td>
<td>yes (1)</td>
</tr>
<tr>
<td>2.1</td>
<td>2.0+1.6.1</td>
<td>2001</td>
<td>PSF</td>
<td>no</td>
</tr>
<tr>
<td>2.0.1</td>
<td>2.0+1.6.1</td>
<td>2001</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.1</td>
<td>2.1+2.0.1</td>
<td>2001</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.2</td>
<td>2.1.1</td>
<td>2001</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.2</td>
<td>2.1.1</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.1.3</td>
<td>2.1.2</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.2.1</td>
<td>2.2</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.2.2</td>
<td>2.2.1</td>
<td>2002</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.2.3</td>
<td>2.2.2</td>
<td>2003</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.3</td>
<td>2.2.2</td>
<td>2002-2003</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.3.1</td>
<td>2.3</td>
<td>2002-2003</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.3.2</td>
<td>2.3.1</td>
<td>2002-2003</td>
<td>PSF</td>
<td>yes</td>
</tr>
<tr>
<td>2.3.3</td>
<td>2.3.2</td>
<td>2002-2003</td>
<td>PSF</td>
<td>yes</td>
</tr>
</tbody>
</table>
Note: 1. GPL-compatible doesn’t mean that we’re distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don’t.

2. According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido’s direction to make these releases possible.

**Terms and Conditions for Accessing or Otherwise Using Python**

**PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2**

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSFhereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided
herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

**BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0**

**BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1**

1. This LICENSE AGREEMENT is between BeOpen.com (“BeOpen”), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization (“Licensee”) accessing and otherwise using this software in source or binary form and its associated documentation (“the Software”).

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the
BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python” logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

**CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1**

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use
Python 1.6.1 alone or in any derivative version, provided, however, that CNRI’s License Agreement and CNRI’s notice of copyright, i.e., “Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved” are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI’s License Agreement, Licensee may substitute the following text (omitting the quotes): “Python 1.6.1 is made available subject to the terms and conditions in CNRI’s License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013.”

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an “AS IS” basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia’s conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant
permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the “ACCEPT” button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
WiX

Copyright 2006-1015 Microsoft
Version 3.9
Microsoft Reciprocal License (Ms-RL)

Description

WiX Toolset License
The WiX toolset is released under the Microsoft Reciprocal License (MS-RL). A reciprocal license is used to ensure that others who build on the effort of the WiX community give back to the WiX community. Specifically the license changes and improvements to the WiX toolset must be published using the same license.

Sometimes the reciprocal license is incorrectly interpreted to also apply to bundles, packages, custom actions built using the WiX toolset. The Outercurve Foundation has provided this statement to clarify:

The WiX toolset (WiX) is licensed under the Microsoft Reciprocal License (MS-RL). The MS-RL governs the distribution of the software licensed under it, as well as derivative works, and incorporates the definition of a derivative work provided in U.S. copyright law. OuterCurve Foundation does not view the installer packages generated by WiX as falling within the definition of a derivative work, merely because they are produced using WiX. Thus, the installer packages generated by WiX will normally fall outside the scope of the MS-RL, and any of your source code, binaries, libraries, routines or other software components that are incorporated in installer packages generated by WiX can be governed by other licensing terms.

Microsoft Reciprocal License (MS-RL)
This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software.

Definitions
The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law.
A "contribution" is the original software, or any additions or changes to the software.
A "contributor" is any person that distributes its contribution under this license.
"Licensed patents" are a contributor's patent claims that read directly on its contribution.

**Grant of Rights**

(A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create.

(B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.

**Conditions and Limitations**

(A) Reciprocal Grants- For any file you distribute that contains code from the software (in source code or binary format), you must provide recipients the source code to that file along with a copy of this license, which license will govern that file. You may license other files that are entirely your own work and do not contain code from the software under any terms you choose.

(B) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks.

(C) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically.

(D) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software.

(E) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(F) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.
Xdan-datetimepicker

Version 2.5.3

JQuery plugin for some drop down lists in Vertica Management Console (MC).

MIT License (MIT)

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
**Xerces**

NOTICE file corresponding to section 4(d) of the Apache License, Version 2.0, in this case for the Apache Xerces distribution.

This product includes software developed by The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following: Software copyright © 1999, IBM Corporation., http://www.ibm.com.
xml-commons

Copyright © 2012

http://www.apache.org/licenses/

Version 2.0, January 2004

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the
purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such
third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
yjal

Copyright (c) 2007-2011, Lloyd Hilaiel

http://github.com/lloyd/yajl

Copyright © 2004-2013 by Internet Systems Consortium, Inc. ("ISC")

Copyright © 1995-2003 by Internet Software Consortium

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Zlib

This is used by the project to load zipped files directly by COPY command. www.zlib.net/
zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.3, July 18th, 2005

Copyright © 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the
authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial
applications, and to alter it and redistribute it freely, subject to the following restrictions:

- The origin of this software must not be misrepresented; you must not claim that you wrote
  the original software. If you use this software in a product, an acknowledgment in the
  product documentation would be appreciated but is not required.

- Altered source versions must be plainly marked as such, and must not be misrepresented as
  being the original software.

- This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu
Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu