White Paper

Analytics & Big Data

Why All Column Stores Are Not the Same

Twelve Low-Level Features That Offer High Value to Analysts

VERTICA | MICRO FOCUS

Table of Contents

There is a common fallacy that columnar store databases are becoming commoditized and that many vendors are filling in the checkbox for column database. While true that there are quite a few databases who started out as a row store and now can reportedly store data in columns, many lack some of the critical capabilities needed for optimal performance. Using a row store engine to store data in columns is quite different than having a native column-store.

In this white paper, we'll discuss twelve critical capabilities for a column store database. From compression techniques to optimization strategies, the detail on how your columnar database works and how the inner workings impact usage are crucial for project success.

Compression

Understand how much space is saved when compressing data, leading to more data on fewer servers and deployment savings.

One of the huge advantages of storing a column of data is in the efficiency of compression. If you have a lot of repeated values in a column, say stock names or items that have been selected from a drop-down menu, RLE and LZH compression will be quite efficient in compressing the data. Numbers in a column are a different matter. Vertica uses integer packing for integers, for example. When you compress efficiently, based on the data that is stored in the column, it lowers the amount of time it takes to push the data through the slowest part of your computer, namely your I/O bus.

In both native columnar databases and pseudo-columnar databases, you probably have some form of compression, but the intelligence behind the compression and the variety of compression algorithms is a key difference. There are nine types of encoding/compression listed in the latest Vertica documentation, each geared toward efficiency for common data types, or you can select Auto.

When you look across solutions in the column store market, most have fewer algorithms and therefore store data less effectively than possible. For example, Hadoop solutions primarily use Snappy, and Zlib compression for ORC and Parquet data. As data volumes grow, the difference in the number of nodes you need to run your analytics is significant because of the differences in columnar compression.

Early and Late Materialization

Determine how the solution accesses and loads data during the query with performance in mind.

Early and Late Materialization for a column store describes when you have to decompress data. This has a huge impact on performance. Depending on the query, you mostly want to keep the data compressed and act on compressed data as much as possible. You only want to decompress it when you have to because buffering and acting on compressed data is more efficient. Typically, the pseudo-columnar solutions rely on early stage materialization, so that once you reference the compressed column, it decompresses it to perform the query. The late stage materialization of a proper column store lets you act on compressed

data and only decompresses for presentation. You'll see the lack of late stage materialization on some row stores that later added column capability.

When you materialize data has a big impact on performance. You can imagine that if you materialize too much data, too soon, it will become a burden on memory management, network traffic, concurrency and more. When choosing a columnar database, make sure that materialization is very efficient.

Predicate Pushdown

Understand how more complex queries are processed in order to determine the database's efficiency.

The database's capability for predicate pushdown can have a big impact on performance and is linked to materialization. If I ask SQL to give me a list of customers WHERE the last name begins with "B", the predicate should be considered in the query plan, otherwise the system will spend a lot of time and resources pulling up last names that don't start with B. It should be able to optimize and know that the customer whose last name begins with B exist mostly in one or two nodes and therefore it's unnecessary to pull out the full list into memory and eliminate the ones that don't meet this condition.

When your solution lacks predicate push-down, it has a similar impact to improper materialization. Memory management, network performance and concurrency issues will occur causing queries to fail or run slowly. You'll see a lack of predicate pushdown most often in the Hadoop world, although they are adding this in many solutions. There's a descriptive section of the Vertica manual that talks more about predicate pushdown and Hadoop file formats here.

Projections and Live Aggregate Projections

Understand how optimizations can boost query performance and the level of management needed to keep them up to date.

What happens if I load my data into my column store and it still doesn't meet my service level agreement with my business stakeholders? It's still not fast enough. Do I have any place to go? Because compression, predicate pushdown and late stage materialization are so efficient in Vertica, you may want to keep copies of some of your data around so that it can be pre-optimized for different types of common queries. In the RDBMS world, materialized views are similar to projections.

For projections, the initial database might be a list of customers as they join the customer list, while a copy might be sorted by last name or region. Then, when you ask for a report by region, the predicate push-down and materialization features kick in and get right to the data you need. If your reports rely on a customer count, you shouldn't have to always ask the database to count it all repeatedly. Instead, you use aggregate projections to keep a running tally. If I want to know the top 10 customer deals this quarter, I should be able to keep a running tally without have to look through the entire database. This is why projections and live aggregate projections are so important in a columnar database.

With other solutions in the market, it's possible to create a copy of the data and manually manage these views as data stores. However, since many of the solutions on the market don't have these optimizations, you're going to have to code it and manage the data copies much more carefully.

Flattened Tables

Compare the method used for boosting JOINs and how the solution can simplify JOIN queries.

So what about optimizing JOINs in a columnar database? What optimizations exist for a columnar store to do JOINs? Normalized database design often uses a star or snowflake schema model, comprising multiple large fact tables and many smaller dimension tables. Queries typically involve joins between a large fact table and multiple dimension tables. These queries can incur significant overhead, especially as the analytical needs of the organization change but the database design remains the same.

Moreover, it's common for database schemas to need to be changed. This becomes clear when you ask yourself a simple question—are you running the same reports this year as you were just one year ago? What about two years ago? Schemas are designed to be optimized for some form of reporting, but the data changes, the needs of analysts change and your business changes. When this happens, you can choose to refactor your database, or you can use a feature like flattened tables.

Vertica flattened tables are denormalized tables that that get their values by querying other tables. To the analyst, it appears that all of the columns needed are in one table. What is actually happening is that Vertica can create a copy of the data in a flattened schema and still be very efficient in storage requirement. Again, using the efficiencies of compression, late stage materialization and predicate push-down, these normalized tables have two interesting benefits: 1) It makes it easier for the business analyst to write simple queries instead of JOINs, and; 2) performance is faster than a normalize JOIN in many cases.

Compressing the Indexes is Not Columnar

Understand exactly what data is compressed its impact on performance.

In standard relational databases, indexes are used to quickly locate the data you need for the query. Of course, you have to spend resources keeping them up to date as new data comes in. This is another reason why columnar databases are so much more efficient than a row store database.

In a columnar database, there should be no indexes because you know exactly where the data is stored for the columns, plus you have predicate pushdown so that subsets of columns are also easily located. If you need to optimize further, you can turn to projections and flattened tables. That's why it's puzzling to see that indexes are still referenced in certain columnar solutions.

Upon further investigation, you may find that some vendors are compressing indexes and calling it columnar. You may find some solutions that can archive data in columnar format, but since the engine is still a row-store engine, you need to index the data. Any columnar database that makes you deal with indexes should be carefully examined.

Architectural Issues—Leader Nodes

Understand if your solution has a single point of failure and weigh the risk of downtime.

Surprisingly, many newer columnar data stores have a major architectural flaw. Certain products have a single point of failure where queries go into "master servers" to create a query plan and dispatch the query to multiple nodes. Although this node is capable of fail-over, it is a trouble spot if that single node goes down.

Don't bet your money on any columnar database with a single point of failure. For a well-architected system, you should be able to query any node. Any node should act as the controller and handle distributing the work across the cluster.

Memory vs Disk Drive

Leverage lower cost commodity servers rather than high-end memory-heavy ones and improve your total cost of ownership.

It's well known that RAM memory is fairly efficient in transferring data when compared to hard disk drive. If you could get all of the data you need to analyze into memory, the query should run fast. Some columnar databases are big on this principle. They are primarily in-memory database and recognize that if you store the data in columns rather than rows, you can compress with greater efficiency and get more data into RAM. In-memory databases are mostly columnar for this reason.

However, databases like this take a huge performance hit when they run out of in-memory storage and analysis spills over to disk. In our testing with products like Spark, queries are fast until you hit this limit. If you want to test this limitation, you have to test your in-memory database with a LOT of data, more data than you have memory, and see how it performs.

A better architecture is to use the memory available and to optimize disk access as much as possible. Instead of assuming in-memory, why not establish a system of very fast disk access and the best utilization of memory available to cache data? Focus the database on its capability to limit the data it has to access from disk with compression, predicate pushdown and late materialization.

Make sure you can run concurrent queries without fail, while also being able to set relative importance. In doing so, give better access to more analytics for all.

Memory Management

One of the most difficult tasks to do is to manage memory for more complex SQL queries. The TPC-DS benchmark queries contain some good examples of analysis that might take a lot of memory, often referred to as "long running queries". Newer column stores simply don't have the maturity to manage memory properly during a long running query. This is what happens when your query runs for 15 hours then fails. Memory management is immature in your solution.

In our benchmarks, immature solutions have a particularly hard time with long-running queries when you have concurrent queries. So, if you send your solution 5 quick queries, 2 medium queries and 1 long query, is it mature enough to manage memory during the test? Can you tell your solution that that some of the quick and medium queries are important and we need them fast, while the rest of the queries can take their time? Our benchmarks show that not every solution has mature mixed workload management. Some of the solutions on the market and in the open source community can't finish all the queries when sent simultaneously.

The key to a mature analytics platform is having a mature workload management system. Vertica's resource management capability allows diverse, concurrent workloads to run efficiently on the database. For basic operations, Vertica pre-configures the built-in general pool based on RAM and machine cores. You can customize the General pool to handle specific concurrency requirements. You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally assign each database user to a specific resource pool, which controls memory resources used by their requests.

In the open source world, you can accomplish this, but you need some combination of Hive, Tez, Solr or Spark combined with Ambari or Zookeeper for operational control and perhaps Yarn for data management. If you want that to be secure, you should bring in Ranger or Apache Sentry. Or you can just do it all in Vertica.

Understand if the database was developed as columnar from the beginning, or as an afterthought.

Columnar Core

It's somewhat common to see row store engines used for column store purposes. Keep in mind that adding column store to a row store engine is not a simple task. There will be efficiencies in building a database to be columnar from the ground up, as we've done with Vertica.

There are also row stores that have later added some columnar capabilities. Solutions from Teradata, Greenplum, Oracle Exadata and others fall into this category. In the case of Oracle, column store is used for archiving and backup so that older, colder data fits tightly in a server. In Oracle Exadata, columnar compression is an effective way to archive data for later retrieval, but the hot data still remains in a row store.

Oracle provides a solution called Oracle's Hybrid Columnar Compression, which primarily uses columnar storage and compression for a hot backup. Teradata and Greenplum take the data you've selected as part of a query and decompressed into rows and then processes the query using their conventional oriented database engines. There's some speed improvement possible in this, but it's primarily early stage materialization without predicate push-down.

There are highly scalable databases that are sometimes confused with column stores but have none of the columnar capabilities for extreme speed and storage. Netezza was a great example of this type of database because it was MPP, but not a true native column store. Therefore, you'll notice that many of the features of a column store database like predicate push-down and materialization don't apply.

Depth of Analysis

Determine the depth of analytical functions so that you can run all the analysis you need without heavy coding.

Speed and query performance aren't the only measure of a modern database. Many of the new solutions on the market focus on the speed and performance of columnar, but lack the in-depth analytics offered by Vertica. Some solutions won't complete the ANSI SQL benchmarks like TPC-DS. Functions like geospatial and time-series analysis might be important to your analytics. If so, some offerings require that you wrangle additional solutions and that you move the data back and forth between them. Others may require extensive coding in Scala, java or python in order to accomplish the task.

A great example of this is time series data. If an IoT device is sending you information consistently, then for whatever reason stops, this can become problematic for your analytics. Without functions for time series analysis, you would have to curate the data to fill in missing values for fear of inaccurate analytics. Functions like gap analysis and missing value imputation can speed the process greatly.

Assess what type of analysis you want to perform and check to make sure that the database supports it. Since some databases are immature, they may lack the capability and open your team up for extra grunt work down the road.

Vertica provides many advanced SQL-based functions from graph analysis to triangle counting to Monte Carlo simulations to geospatial and more. Vertica's in-database machine learning capabilities allow users to take advantage of Big Data while simplifying and speeding up their predictive analytics processes to make better informed decisions, compete more effectively, and accelerate time-to-insight. From data prep to deployment, Vertica supports the entire machine learning process, and allows models to be deployed across Vertica clusters, a key requirement for solutions that embed Vertica as their analytics engine.

Understand how you're going to leverage external data without having to load it and/or migrate off of the platform if needed. Make sure the database supports information lifecycle management.

Openness and Lock-in

What if you want to use data that's sitting outside the database? The file formats ORC and Parquet are common column store formats. Can you leverage those formats without having to copy the data? Some databases like Vertica have functions for reading external tables without having to load them in the database. Some solutions can also apply schema on read to file formats like JSON while others require an external tool like an ETL tool.

Analysts and database administrators don't want to be locked-in to one cloud, or one solution. More than that, they want to be able to analyze data that's already stored in columnar format as it sits. If data becomes less critical, they may want to tier it off to a more cost effective storage layer, thus supporting information lifecycle management.

Be wary of columnar databases in the market that lock you in to specific infrastructure. Snowflake and Redshift are great examples of this. You're locking yourself into the Amazon cloud with very few paths for getting out.

Final Thoughts

If you're evaluating database features, don't just take it for granted that a columnar database is a commodity. There are now more databases than ever before that claim to be column stores. However, your performance will vary greatly, based on the differences outlined above. Don't be afraid to ask your vendor about these features. If you work with an analyst, make sure he or she understands the intricacies of a column store database before you take their recommendation for a solution. You may find that they have no clear grasp of the topic.

Selecting the right analytical database requires more than a simple "column store" check box. It's about identifying a mature, performant solution that:

- Doesn't require expensive memory-heavy servers (or GPUs) to be performant
- Speeds up your queries when you have lots of data or many users
- Can handle mixed workloads and different user expectations
- Doesn't lock you in by making it difficult to get data out
- Is easy to manage and always optimized

It is these characteristics that have the biggest impact on your team.

Vertica

Vertica provides a powerful analytics platform based on columnar storage. At its base, Vertica is a SQL database that was purpose-built for advanced analytics against big data. It leverages revolutionary massively parallel, columnar storage optimizations to deliver high performance with very complex analytic queries against multi-terabyte datasets.

While SQL is the primary query and analysis language, Vertica also supports Java, Python, R, and C. The analytical functions are vast and include time-series, data preparation, geospatial and other advanced functions. What's more, Vertica can perform in-database machine learning like linear regression, logistic regression, SVM, K-means, Naïve-Bayes and more to deliver predictive analytics. You can stop copying data out of your data warehouse when you want to do predictive analytics. Vertica leverages the MPP cluster and optimizations when training and performing machine learning analytics.

In a modern enterprise architecture, gaining access to data that's stored in multiple locations is crucial to delivering analytics. Vertica allows you to perform analysis on data that is sitting in HDFS (Apache Hadoop). It doesn't really matter if the data is stored in your database or in ORC or Parquet formats, Vertica can perform analysis on it without having to first copy it. Furthermore, the Vertica Flex table feature enables users to define and apply schema during query and analysis, thereby handling formats like JSON.

In most cases, Vertica is less expensive to license than legacy database solutions and is a natural fit when updating those systems. ETL and data visualization tools will work the same, only faster, while you save on costs on tuning indexes and optimizing your legacy systems. When it's time to move to cloud, you can take your Vertica license and deploy on Amazon, Microsoft Azure or Google clouds. Simply specify one of these locations in our management console and deploy.

Download Vertica's free Community Edition today at: **vertica.com**

Learn More At
www.vertica.com

In most cases, Vertica is less expensive to license than legacy database solutions and is a natural fit when updating those systems.

Additional contact information and office locations:
www.vertica.com

www.vertica.com

VERTICA | MICRO FOCUS®