



## User Defined Extensions in Vertica

Vertica's broad array of User Defined capabilities (Functions, Transforms, Aggregates, Analytics, and Loading) bring the power and flexibility of procedural code closer to the data – be it structured, semi-structured, or unstructured – fully leveraging the parallel compute environment of the Vertica platform. User Defined Extensions run in-process for maximum efficiency, or fenced for additional control. In either mode, Vertica's user interface makes it easy to deploy and use procedural extensions, encouraging maintainable operational practices and promoting code reuse.

## Procedural vs. Declarative

For many people, it is easier to approach a solution procedurally, thinking of the steps required to break the task into a sequence of actions. By contrast, SQL is a declarative language, where the operator states the required outcome, and the database develops an optimal procedure and efficiently computes the answer. Vertica's Optimizer is no exception – it understands how the data is distributed about the cluster, the most cost effective join order, and generates highly efficient query plans (the steps in a procedure). Nevertheless, some operations are difficult or tedious to express in SQL, especially when the inputs are unstructured or semi-structured. For cases where procedural language is more natural, User Defined Extensions are the natural choice.

## Code Maintenance and Reuse

SQL requires a schema to organize the data, while procedural code allows the author the flexibility to manipulate the data arbitrarily. Each has advantages. User Defined Extensions have clearly defined inputs and outputs, which Vertica presents in searchable tables. Vertica automates physical design and optimally marshals arbitrarily large input and output streams. This creates opportunities for Vertica's Optimizer to efficiently schedule procedures into the most cost effective phase of a query plan, exploiting parallelism without compromising correctness of results. User Defined extensions scale to fit the data footprint, while the author is free to focus on the solution. Hence, the organized nature of a relational system encourages maintainability and code reuse, with uncompromised scale.

## Log Parsing – A Practical Example

One of the examples within the Vertica SDK (included in the base Vertica distribution) is an Apache Web Log parser. This provides a simple way to demonstrate the marriage of procedural code with SQL, as well as demonstrating how easy it is to bring parallel execution close to data.

Vertica distributes procedures to each node in the cluster, managing version control and ensuring that if nodes are down, they will get the updated procedures when they are next online. Registering a procedure is simple:

```
CREATE LIBRARY TransformFunctions AS <path_to_file>;
CREATE TRANSFORM FUNCTION ApacheParser
  NAME 'ApacheParserFactory'
  LIBRARY TransformFunctions;
```

Recall that Vertica automates physical design. A developer who is familiar with procedural languages doesn't need to learn DBA skills to leverage the power and parallelism of a Vertica cluster. Loading semi-structured data into Vertica is easy:

```
CREATE TABLE raw_logs (data VARCHAR(64000));
COPY raw_logs FROM <path_to_file>;
```

Now that we have our parser library loaded and our raw log data available, we're ready to run arbitrary queries as if the log data were structured. Recall that registering our library informs Vertica of the inputs and outputs of our User Defined Extension. We can either create a table as select (CTAS) through the ApacheParser, or skip the intermediate step and query the raw\_logs directly through the ApacheParser as if it were structured data – in either case, column names are provided by the User Defined Transform, e.g.:

```
SELECT COUNT( user_agent ), user_agent
  FROM (SELECT ApacheParser(data)
        OVER (PARTITION BY 1) FROM raw_logs) AS x
 WHERE response_code = '304'
  GROUP BY 2 ORDER BY 1 DESC LIMIT 5;
```

count	user_agent
2752	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
1898	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0) Gecko/20100101 Firefox/8.0
803	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
584	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0.1) Gecko/20100101 Firefox/8.0.1
493	Mozilla/5.0 (Windows NT 5.1; rv:8.0) Gecko/20100101 Firefox/8.0

In this case, we're ranking the browsers according to a specific response code, but it would be just as easy to query on any other relation that may interest us. This type of function could easily serve as input to Vertica's built in sessionization capabilities, so that we could assign a Session ID to each individual visiting our web site, and further use Pattern Matching and Event Series Joins to identify and rank the click paths that lead to unique outcomes. Our scope here is primarily the User Defined Extensions, rather than the built-in, so we'll continue with our example.

## Window Functions and Parallelism

A keen reader may note the PARTITION BY clause in the above SQL. Vertica draws a distinction between User Defined *Functions* (which take one row of input for each row of output they produce) vs. User Defined *Transforms* (which take an arbitrary number of rows for input, and emit an arbitrary number of rows). The distinction is not a matter of performance; both Extension types have a setup phase, independent from iteration, so they need not be called multiple times to process multiple lines. Instead, the PARTITION BY clause is specific to Transforms, and tells Vertica which rows belong together within an *instance of the procedure* the transform applies.

Consider a simple aggregate transform [e.g. computing a weighted average] as an example. By partitioning by "symbol" in a stock ticker table, each instantiation of our weighted average function operates over the prices corresponding to a particular ticker symbol. Vertica's Optimizer is aware that our table contains multiple symbols, so Vertica is able to scale the throughput by running multiple threads of this transform.

Our ApacheParser is a Transform, but internally each line it processes is handled independently. Therefore, we can safely partition the collection of input rows arbitrarily, and still get correct output rows. Therefore, we can safely create an arbitrary

number of partitions, and Vertica will then have opportunities to run multiple threads of our Transform in parallel, e.g.:

```
SELECT COUNT( user_agent ), user_agent
FROM (SELECT ApacheParser( data )
OVER (PARTITION BY HASH(SUBSTR( data , 1 , 20 ))
FROM raw_logs) AS x
WHERE response_code = '304'
GROUP BY 2 ORDER BY 1 DESC LIMIT 5;
```

This partitioning tells Vertica to look at the first 20 characters of each raw log line, and use a hash of those characters to group log lines into a partition. Each thread of the ApacheParser will consume one or more partitions of the raw\_logs. Of course the result of our function is identical to the above example; we've simply allowed Vertica the opportunity to take advantage of more CPU cores across our cluster.

On review of the above query, Vertica's Database Designer recommends (and optionally automatically implements) the following change to the raw\_logs table:

```
CREATE TABLE raw_logs (data VARCHAR(64000))
SEGMENTED BY HASH(SUBSTR(data,1,20)) ALL NODES;
```

This optimization applies the same distribution to the data that Database Designer observed in a query against this table. The results:

- Lines of raw log data are optimally distributed across the cluster nodes as they are loaded into the database
- Vertica's Optimizer now has the opportunity to schedule threads of ApacheParser on each cluster node, and feed each thread with log lines that are local to that node. This minimizes the network traffic between nodes, reducing the cost of the query.

## Simple. Fast. Efficient.

Vertica's User Defined Extension framework combines the intuitive nature of procedural code with the maintainability and operational efficiencies which distinguish Vertica as the leading Analytic Database Platform. Structured, semi-structured, and unstructured data become directly addressable within Vertica. Just like SQL operations, User Defined Extensions work hand-in-glove with Vertica's Database Designer, column aware Optimizer and "scale-out" architecture to seamlessly scale both procedural and declarative execution to meet the requirements of any size data footprint or analytic workload.



Vertica, An HP Company 8 Federal Street, Billerica, MA 01821 +1.978.600.1000 TEL +1.978.600.1001 FAX [www.vertica.com](http://www.vertica.com)

© Vertica 2012. All rights reserved. All other company, brand and product names contained herein may be trademarks or registered trademarks of their respective holders.

### About Vertica

[Vertica, an HP Company](http://www.vertica.com), is the leading provider of next-generation analytics platforms enabling customers to monetize all of their data. Vertica's elasticity, scale, performance, and simplicity are unparalleled in the industry, delivering 50x-1000x the performance of traditional solutions at 30% the total cost of ownership. Vertica powers some of the largest organizations and most innovative business models globally including Zynga, Groupon, Twitter, Verizon, Guess Inc., Admeld, Capital IQ, Mozilla, AT&T, and Comcast.