

# Visual Guide to Data Loading with HPE Vertica

## Choosing a Load Method

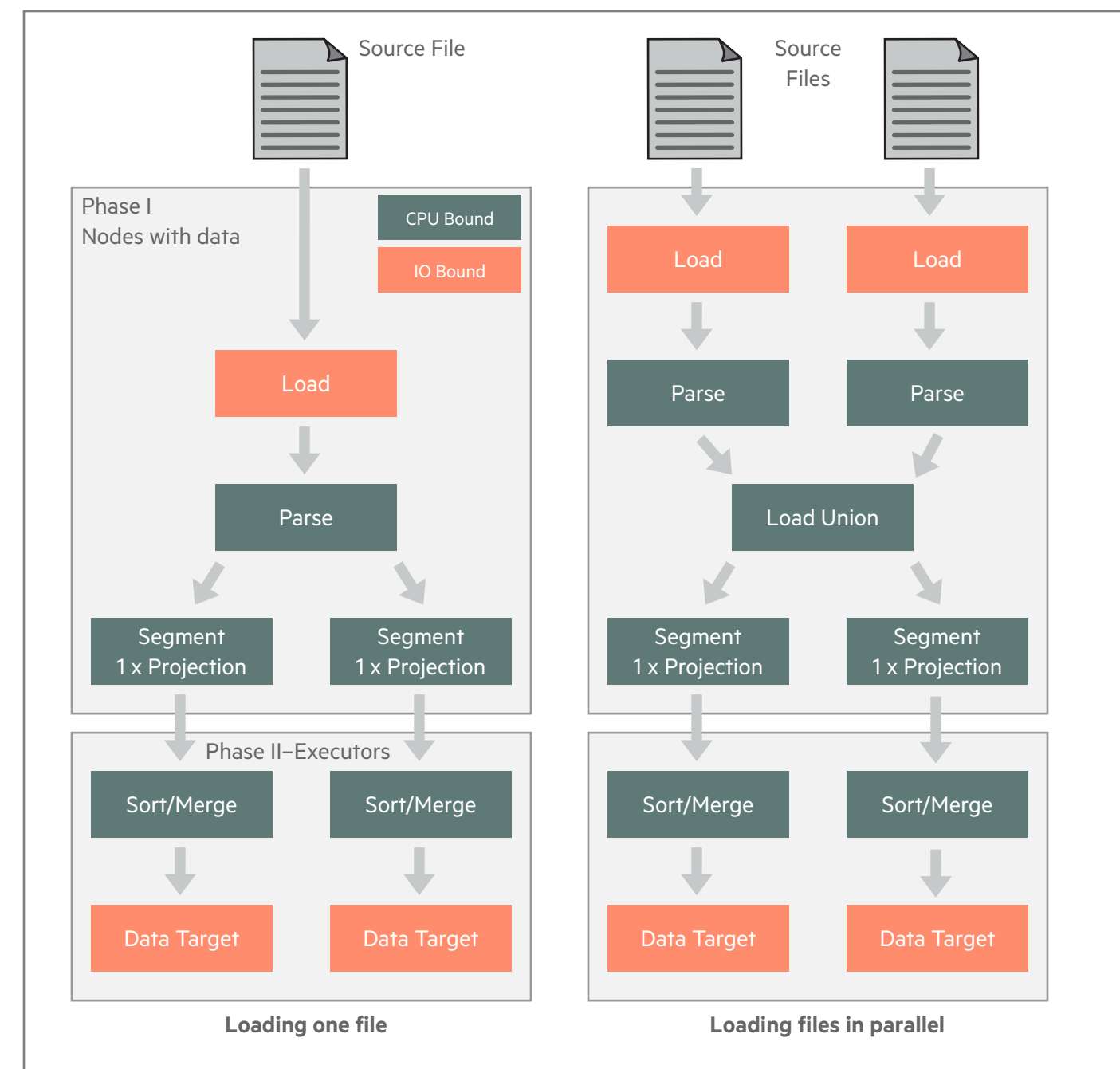
Depending on what data you are loading, the COPY statement includes several load options.

Load Method	Description
<b>AUTO</b>	This is the default load method. If you do not specify a load option explicitly, COPY uses the AUTO method to load data into WOS (Write Optimized Store) in memory. The default method is good for smaller bulk loads (< 100MB). Once the WOS is full, COPY continues loading directly to ROS (Read Optimized Store) on disk. ROS data is sorted and encoded.
<b>DIRECT</b>	Use the DIRECT parameter to load data directly into ROS containers, bypassing loading data into WOS. The DIRECT option is best suited for large data loads (100MB or more). Using DIRECT to load many smaller data sets results in many ROS containers, which have to be combined later.
<b>TRICKLE</b>	Use the TRICKLE option to load data incrementally after you complete your initial bulk load. Trickle loading loads data into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. Use this option only when you have a finely-tuned load and moveout process at your site, and you are confident that the WOS can hold the data you are loading. This option is more efficient than AUTO when loading data into partitioned tables.
<b>COPY PARSERS</b>	By default, COPY uses the DELIMITER parser to load raw data into the database. Raw input data must be in UTF-8, delimited text format. COPY parsers include: <ul style="list-style-type: none"> <li>DELIMITED</li> <li>NATIVE BINARY</li> <li>NATIVE VARCHAR</li> <li>FIXED-WIDTH</li> <li>ORC</li> <li>PARQUET</li> </ul>

## Understanding how the COPY command works

Vertica's bulk loader (COPY) will automatically create separate worker threads to process a single file. This means that a file will be read very quickly by multiple, parallel readers. In addition, Vertica will try to divide the work across all of the CPU cores and nodes, in the cluster, for multiple files. This significantly reduces the task of optimizing most bulk/trickle load tasks.

However, in some instances, it may be wise to manually divide the COPY workload across all of your nodes. In the case, put the data files where they are accessible from the target nodes. Then run multiple, parallel COPY statements for each node. A good place to start is to create a single COPY running for each node. You may add additional COPY statements, for each node, to improve bulk loading throughout. However, be cautious with the number of parallel COPY statements executing by monitoring system resource (CPU, memory, disk I/O, and network) utilization. You don't want to create a lot of resource contention.



## Bulk Loading Data

There are different methods for bulk loading data into a Vertica database using the COPY statement. In its basic form, use COPY as follows:

```
COPY target-table FROM data-source
```

The COPY statement loads data from a file stored on the host or client (or in a data stream) into a database table. You can pass the COPY statement many different parameters to define various options such as:

- The format of the incoming data
- Metadata about the data load
- Which parser COPY should use
- Load data over parallel load streams
- How to transform data as it is loaded
- How to handle errors

Vertica's hybrid storage model provides a great deal of flexibility for loading and managing data.

For more information on data loading refer to the Vertica documentation online at [myvertica.com/docs](http://myvertica.com/docs).

## Column Delimiter Options

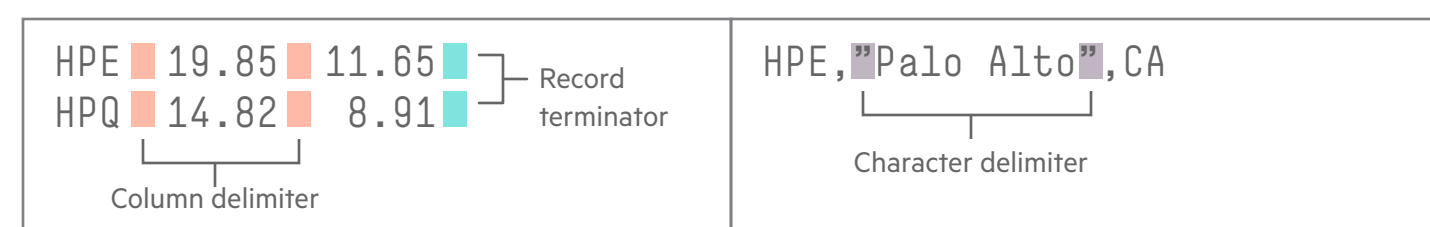
The default COPY delimiter is a '|' vertical bar. The DELIMITER is a single ASCII character used to separate columns within each record of a file. Between two delimiters, COPY interprets all string data in load files as characters.

```
Change default to comma delimiter.
COPY mytable FROM STDIN DELIMITER ',';

Specify TAB character column delimiter.
COPY mytable FROM STDIN DELIMITER 't';

Specify any ASCII character as delimiter character.
COPY mytable FROM STDIN DELIMITER 'x1e';

Specify any ASCII character as record delimiter.
COPY mytable FROM STDIN DELIMITER ','
RECORD TERMINATOR 'x1e';
```

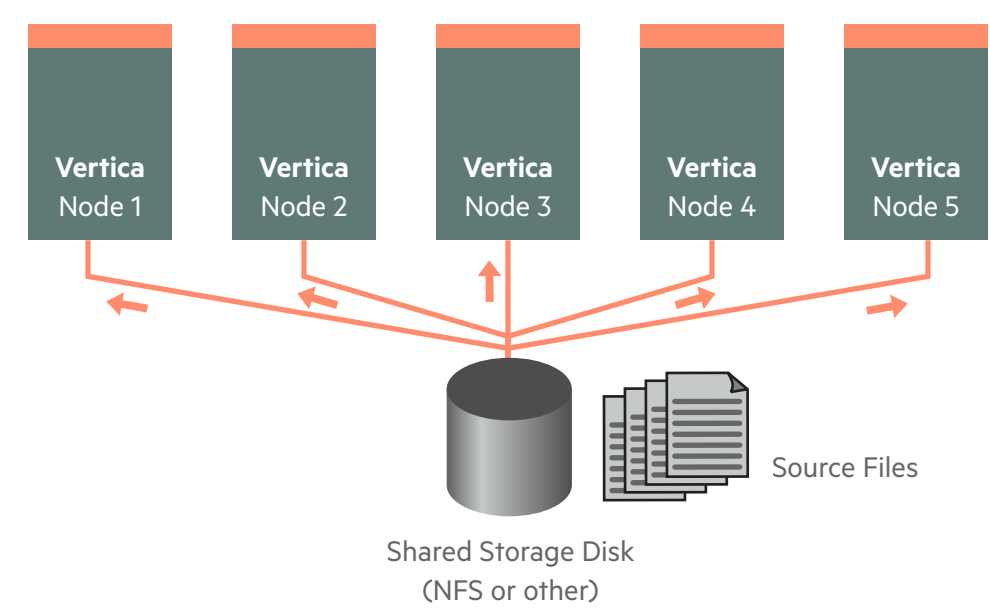


## Data Loading by Example

For loading initial data sets, it is important to review the rejected rows before loading many terabytes of data. Try to correct the rejected rows as much as possible for subsequent data loads. Even a handful of rejected row may cause the COPY statement to slow down when compared to zero rejected rows. Corrections are typically done by either modifying the source data files (typically by creating an external (Shell, Python, etc.) script to fix a specific formatting problem) or by modifying the COPY statement parameters to understand/accept the rejected row.

```
Load Compressed File
Load a single, gzipped CSV (with pipe delimiter) file on a single node:
COPY schema-name.target-table FROM '/path/on/node/file.gz' ON node0001
GZIP DIRECT;
```

```
Load Multiple Files from Shared Device
Load multiple CSV (with tab delimiter) files from a shared storage device (connected to all nodes):
COPY schema-name.target-table FROM '/shaced/path/*.csv' ON ANY NODE
UNCOMPRESSED DELIMITER 't' DIRECT;
```



Using a wildcard with the ON ANY NODE clause expands the file list on the initiator node. This command then distributes the individual files among all nodes, so that the COPY workload is evenly distributed across the cluster.

```
Load Specific Columns
Load specific columns, in a specific order, from a CSV (with comma delimiter) file, on a single node:
COPY schema-name.target-table (c, b, a) FROM '/path/on/node/file.csv' ON
node0001 UNCOMPRESSED DELIMITER ',' DIRECT;
```

```
Skip Lines in Load
Skip a header (first line) in a CSV file. This prevents Vertica from rejecting the line.
COPY schema-name.target-table FROM '/path/on/node/file.csv' SKIP 1 DIRECT;
```

```
Transform Data During Load
Transform data, at load time, using fillers from a CSV (with pipe delimiter) file on a single node. This transforms column C into a TIMESTAMP datatype from a string like '06 Jul 2006'. Any expressions can be applied to transform a column at load time, which saves a lot of post-processing time and effort.
COPY schema-name.target-table (a, b, c,temp FILLER VARCHAR, c AS
TO_TIMESTAMP(c,temp, 'DD Mon YYYY')) FROM '/path/on/node/file.csv' ON node0001
UNCOMPRESSED DIRECT;
```

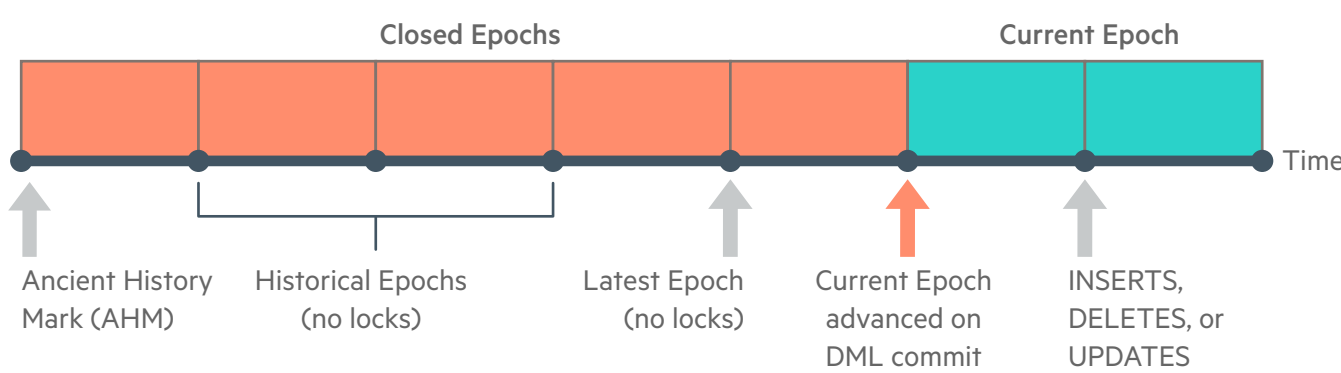
```
Combine Columns
This combines columns A and B into a single VARCHAR:
COPY schema-name.target-table (a,temp FILLER VARCHAR, b,temp FILLER INTEGER, a
AS CONCAT(CONCAT[a,temp, '-'], b,temp::VARCHAR)) FROM '/path/on/node/file.csv'
ON node0001 UNCOMPRESSED DIRECT;
```

## Vertica Transaction Model

**AHM:** The Ancient History Mark epoch prior to which historical data can be purged from physical storage.

**Epoch:** A 64-bit number representing a logical timestamp for data in Vertica. Every row has an implicitly stored column recording the committed epoch.

The epoch advances when the logical state of the system changes or when data is committed with DML operations (INSERT, UPDATE, MERGE, COPY, or DELETE). The EPOCHS system table contains the date and time of each closed epoch and corresponding epoch number.



## Using COPY Interactively

You can also use commands such as the following interactively by piping a text file to vsql and executing a COPY (or COPY FROM LOCAL) statement with the standard input stream as the input file. For example:

```
$ cat myfile.txt | vsql -c "COPY mytable FROM STDIN DELIMITER '|' DIRECT";
$ cat myfile.txt | vsql -c "COPY mytable FROM LOCAL STDIN DELIMITER '|' DIRECT";
```

## Vertica Flexible Tables

Flex tables simplify data loading by allowing you to browse semi-structured data without having to create a schema or column definitions. Flex tables support full SQL functionality. Formats supported include: Avro Data, CEF, CSV, Delimited, and JSON.

```
Source Data Example
The flex table feature loads JSON or delimited files without the need to define each column and data type. In this example, a simple CSV file will be used, create the file as follows and save as /source/cities.csv.
City,State,Zip,Population
Boston,MA,02108,655884
Chicago,IL,60601,2722389
Seattle,WA,98101,668342
Dallas,TX,75201,1281047
New York,NY,10001,8491079
```

```
Create Flex Table and Load the Data
Create the flex table and name it cities_flex. Note, column names and types are not necessary with flex tables.
create flexible table cities_flex();
copy cities_flex from
'/source/cities.csv' parser
fdelimitedparser (delimiter=',');
```

```
Query the cities_flex table, specifying column names from original CSV source file.
select city,state from cities_flex;
city | state
-----|-----
Boston | MA
Chicago | IL
Seattle | WA
Dallas | TX
New York | NY
(5 rows)
```

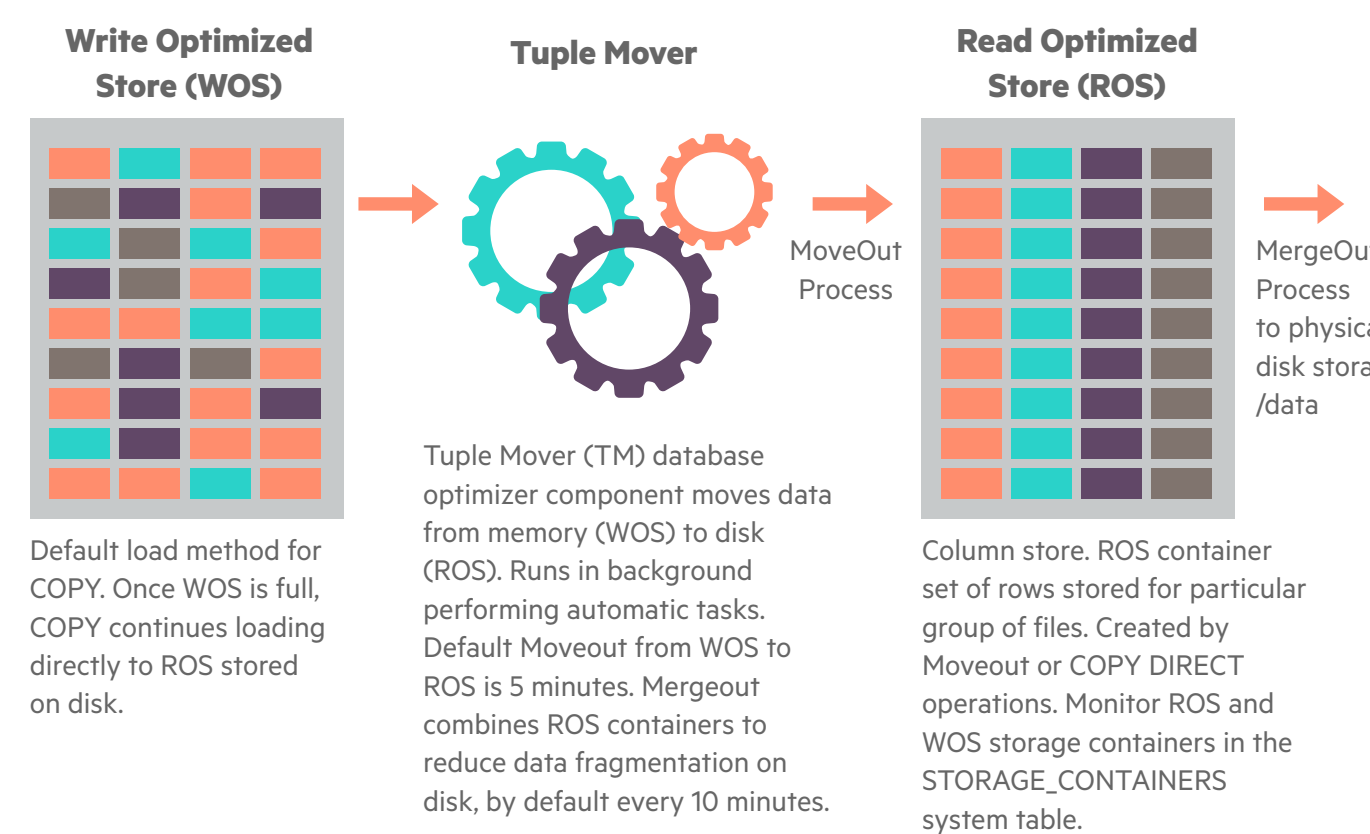
```
To view actual raw contents, run the following:
select * from cities_flex;
```

```
Use the MapToStringO function to view the contents in JSON text format.
select maptostring(...row...) from
cities_flex limit 1;
```

```
Build Flex Table View
Compute the flex table keys and build a view in single statement. The new view name is cities_flex_view.
select
compute_flextable.keys_and_build_view('ci
ties_flex');
```

```
Adjust Data Types
The Zip and Population columns can be updated in from the default varcharO data type to int.
update cities_flex.keys set
data_type_guess = 'int' where
key_name='zip';
update cities_flex.keys set
data_type_guess = 'int' where
key_name='population';
commit;
```

## Vertica Storage Architecture



## Hybrid Storage Architecture

Default load method for COPY. Once WOS is full, COPY continues loading directly to ROS stored on disk.

## User Privileges for COPY

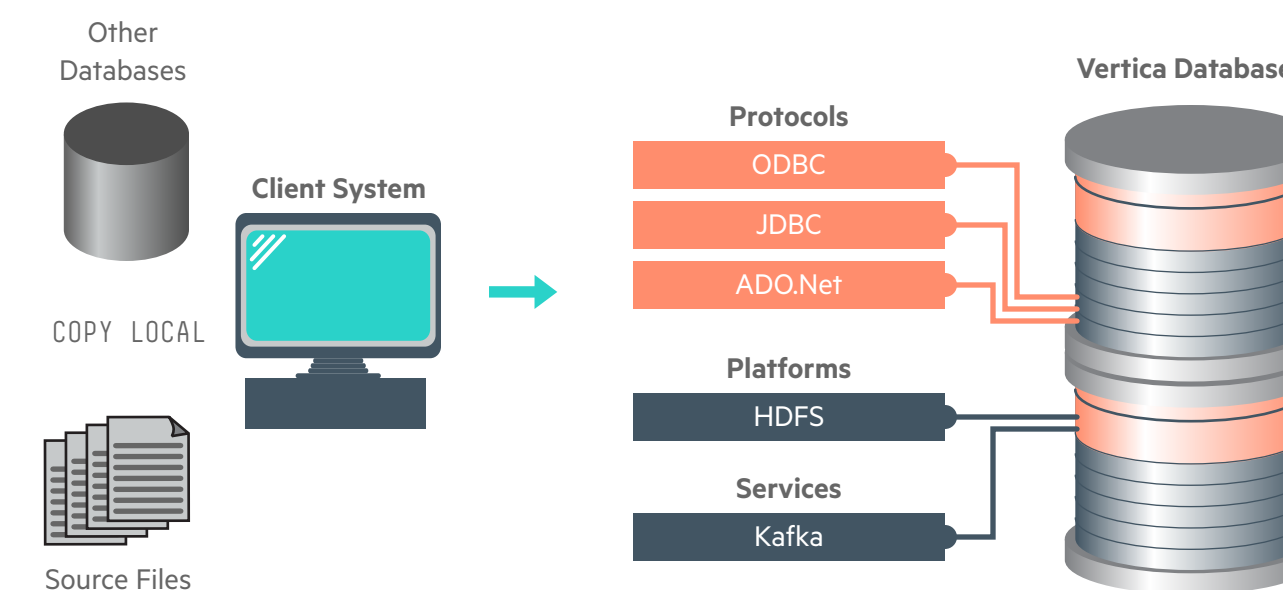
**Privileges**  
You must connect to the HPE Vertica database as a superuser, or, as a non-superuser, have a user-accessible storage location, and applicable READ or WRITE privileges granted to the storage location from which files are read or written to. COPY LOCAL users must have INSERT privileges to copy data from the STDIN pipe, as well as USAGE privileges on the schema.

The following permissions are required to COPY FROM STDIN:

```
INSERT privilege on table
USAGE privilege on schema
```

## Client Connection Methods for Loading

You can load data into your Vertica database through client connections from other databases or sources. Common ETL tools connect to Vertica and useful for data loading.



Client Connection and Loading Best Practices

- Native connection load balancing is a feature built into the HPE Vertica server and client libraries. Load balancing spreads the CPU and memory overhead caused by client connections across the hosts in the database. Native connection load balancing only has an effect when it is enabled by both the server and the client.
- Pushdown optimization (source side, target side, and full)
- Writing bulk data to disk directly with COPY using DIRECT option
- Avoid single row INSERTs for best performance

## Monitoring Loads and Reporting

**Using the LOAD\_STREAMS System Table**  
HPE Vertica includes a set of system tables that include monitoring information. The LOAD\_STREAMS system table includes information about load stream metrics from COPY and COPY FROM VERTICA statements.

```
To view all table columns:
select * from load_streams;
```

**Using the STREAM NAME Parameter**  
Using the STREAM NAME parameter as part of the COPY statement labels COPY streams explicitly so they are easier to identify in the LOAD\_STREAMS system table.

```
To use the STREAM NAME parameter:
COPY mytable FROM myfile DELIMITER '|' DIRECT STREAM NAME 'My stream name';
```

The LOAD\_STREAMS system table includes stream names for every COPY statement that takes more than 1 second to run. The 1-second duration includes the time to plan and execute the statement.

HPE Vertica maintains system table metrics until they reach a designated size quota (n kilobytes). The quota is set through internal processes and cannot be set or viewed directly.

## Other LOAD\_STREAMS Columns for COPY Metrics

These LOAD\_STREAMS system table column values depend on the load status:

```
ACCEPTED_ROW_COUNT
REJECTED_ROW_COUNT
PARSE_COMPLETE_PERCENT
SORT_COMPLETE_PERCENT
```

When a COPY statement using the DIRECT option is in progress, the ACCEPTED\_ROW\_COUNT value can increase to the maximum number of rows in the input file as the rows are being parsed.

If COPY reads input data from multiple named pipes, the PARSE\_COMPLETE\_PERCENT value will remain at zero (0) until all named pipes return an EOF. While COPY awaits an EOF from multiple pipes, it can appear to be hung. However, before canceling the COPY statement, check your system CPU and disk accesses to see if any activity is in progress.

In a typical load, the PARSE\_COMPLETE\_PERCENT value can either increase slowly to 100%, or jump to 100% quickly. If you are loading from named pipes or STDIN, the SORT\_COMPLETE\_PERCENT value is 0. Once PARSE\_COMPLETE\_PERCENT reaches 100%, SORT\_COMPLETE\_PERCENT increases to 100%. Depending on the data sizes, a significant lag can occur between the time PARSE\_COMPLETE\_PERCENT reaches 100% and the time SORT\_COMPLETE\_PERCENT begins to increase. This example sets the VSOL expanded display, and then selects various columns of data from the LOAD\_STREAMS system table:

```
-> \pset expanded
Expanded display is on.
-> SELECT stream_name, table_name, load_start, accepted_row_count,
rejected_row_count, read_bytes, unsorted_row_count, sorted_row_count,
sort_complete_percent FROM load_streams;
-[ RECORD 1 ]-----
stream_name | Fact-13
table_name | Fact
load_start | 2010-12-28 15:07:41.132053
accepted_row_count | 900
rejected_row_count | 100
read_bytes | 11975
input_file_size_bytes | 0
parse_complete_percent | 0
unsorted_row_count | 3600
sorted_row_count | 3600
sort_complete_percent | 100
```