What is WOS in Vertica?

Write Optimized Store (WOS) is a memory-resident data structure for storing INSERT, UPDATE, DELETE, and COPY (without /\*+DIRECT\*/ hints) actions. To support very fast data load speeds, the WOS stores records without data compression or indexing. The WOS organizes data by epoch and holds both committed and uncommitted transaction data

What is ROS in Vertica?

Read Optimized Store (ROS) is a highly optimized, read-oriented, disk storage structure. The ROS makes heavy use of compression and indexing. You can use the COPY...DIRECT and INSERT (with /\*+DIRECT\*/ hints) statements to load data directly into the ROS.

What is Tuple Mover (TM) in Vertica?

    The Tuple Mover (TM) is the database optimizer component of Vertica that moves data from memory (WOS) to disk (ROS), combines small ROS containers into larger ones, and purges deleted data. The Tuple Mover runs in the background. Under ordinary circumstances, the operations performed by the Tuple Mover (TM) are automatic and transparent, and are therefore of little or no concern to the database administrator.

What is Projection?

Optimized collections of table columns that provide physical storage for data. A projection can contain some or all of the columns of one or more tables. A projection that contains all of the columns of a table is called a super-projection. A projection that joins one or more tables is called a pre-join projection.

Table and Projection are physical or logical?

Table:  
    Tables do not occupy any physical storage at all in Vertica.  
  
Projection:  
    Physical storage consists of collections of table columns called projections.

What is K-Safe in Vertica?

Vertica uses the concept of K-Safety for failure recovery. The K value represents the maximum number of nodes in a database that can fail and recover with no loss of data. In Vertica, the value of K can be zero (0), one (1), or two (2). The Physical Schema design must meet certain requirements. To create designs that are K-Safe, Vertica recommends using the Database Designer.

To determine the number of nodes required to meet your K-Safety needs, use the formula: K<N/2

where :

                     K is the K-Safety value.

                     N is the number of nodes.

The value of K can be 1 or 2 only when the physical schema design meets certain redundancy requirements. See [Physical Schema](https://my.vertica.com/docs/4.1/HTML/Master/1248.htm). To create designs that are K-Safe, Vertica recommends that you use the [Database Designer](https://www.blogger.com/null).

By default, Vertica creates K-Safe superprojections when the database has a K-Safety greater than 0 (K>0). When creating projections with the Database Designer, projection definitions that meet K-Safe design requirements are recommended and marked with the K-Safety level. Note the output from running the optimized design script generated by the Database Designer in the following example:

What is Hash-Segmentation and Unsegmented?

Hash-Segmentation:

    Segments a projection evenly and distributes the data across nodes using a built-in hash function. Creating a projection with hash segmentation results in optimal query execution. HP Vertica recommends segmenting large tables.

Unsegmented:

    Automatically replicates the unsegmented projection on each node. To perform distributed query execution, HP Vertica requires an unsegmented copy of each small table superprojection on each node.

What is DBD in Vertica?

The Database Designer is a tool that recommends a physical database design (projections) that provides the best performance to answer your query needs. This is useful because it minimizes the time the DBA spends on physical database tuning and provides the ability to re-design the database incrementally to optimize for changing workloads over time.

You can run the Database Designer before or after a Vertica database has been deployed, and it runs as a background process.

When you use the Database Designer to create a design, the DBA provides the following inputs:

                     Logical schema (CREATE TABLE statements)

                     Sample data

                     A sample set of queries that represent what you'd normally run

                     A K-safety level

What ANALYZE\_STATISTICS will do?

 ANALYZE\_STATISTICS is a DDL operation that auto-commits the current transaction, if any. The ANALYZE\_STATISTICS function reads a fixed, 10 percent of disk contents to aggregate sample data for statistical analysis. To obtain a larger (or smaller) data sampling, use the [ANALYZE\_HISTOGRAM](https://my.vertica.com/docs/7.1.x/HTML/Content/Authoring/SQLReferenceManual/Functions/VerticaFunctions/ANALYZE_HISTOGRAM.htm) function, which lets you specify the percent of disk to read. Analyzing more that 10 percent disk space takes proportionally longer to process but results in a higher level of sampling accuracy. ANALYZE\_STATISTICS is supported on local temporary tables, but not on global temporary tables.

What is Copy Command in Vertica?

     Bulk loads data from one or more files or pipes on a cluster host into a Vertica database. (See [LCOPY](http://my.vertica.com/docs/5.0/HTML/Master/9596.htm) to load from a data file on a client system using ODBC.)

COPY can load data in one of three formats:

* Text with delimiters (the default format)
* Native binary using the NATIVE keyword
* Native varchar using the NATIVE VARCHAR keyword

How to monitor Copy Command?

You can check COPY loads using HP Vertica “functions” and “LOAD\_STREAMS system table”.  
  
**HP Vertica Functions:**  
 **1. GET\_NUM\_ACCEPTED\_ROWS :**  
To get the number of accepted rows, use the GET\_NUM\_ACCEPTED\_ROWS function  
  
Query:  
SELECT GET\_NUM\_ACCEPTED\_ROWS()  
  
**2. GET\_NUM\_REJECTED\_ROWS :**  
To check the number of rejected rows, use the GET\_NUM\_REJECTED\_ROWS function  
  
Query:  
SELECT GET\_NUM\_REJECTED\_ROWS()  
  
**3.  Include the CURRENT\_LOAD\_SOURCE function as a part of the COPY statement:**  
When you include the CURRENT\_LOAD\_SOURCE function as a part of the COPY statement, the input file name or value computed from it, can be inserted into a column.  
  
To insert the file names into a column from multiple source files:  
COPY t (c1, c2, c3 as CURRENT\_LOAD\_SOURCE()) FROM '/home/load\_file\_1' ON  exampledb\_node02,  
'/home/load\_file\_2' ON exampledb\_node03 DELIMITER ',';  
  
 **LOAD\_STREAMS System Table:**  
    HP Vertica includes a set of system tables that include monitoring information, as described in Using System Tables. The LOAD\_STREAMS system table includes information about load stream metrics from COPY and COPY FROM VERTICA statements, so you can query table values to get COPY metrics.  
  
To see all table columns:  
SELECT \* FROM LOAD\_STREAMS;

What is Mergeout in Vertica?

A mergeout is the process of consolidating ROS containers and purging deleted records. Over time, the number of ROS containers increases to a degree that it becomes necessary to merge some of them in order to avoid performance degradation. At that point, the Tuple Mover performs an automatic mergeout, which combines two or more ROS containers into a single container. This process can be thought of as "defragmenting" the ROS.

What is Encoding and Compression in Vertica?

**Data Encoding and Compression:**  
        HP Vertica uses both encoding and compression to optimize query performance and save storage space. However, in your HP Vertica database, they have different meanings.  
  
**Encoding:**         Encoding is the process of converting data into a standard format. In HP Vertica, encoded data can be processed directly, but compressed data cannot. HP Vertica uses a number of different encoding strategies, depending on column data type, table cardinality, and sort order. Encoding increases performance because there is less disk I/O during query execution. In addition, you can store more data in less space. Encoding is not the same as compression.  
  
**Compression:**          Compression is the process of transforming data into a compact format. Compressed data cannot be directly processed; it must first be decompressed. HP Vertica uses integer packing for unencoded integers and LZO for compressible data. Although compression is generally considered to be a form of encoding, the terms have different meanings in HP Vertica.

How to find currently running sessions?

Vertica is having a table **'Sessions'** to maintain currently running sessions information. Using Sessions table we can find the current running sessions information.  
Query: Connect to DB as Admin and run the below statement  
SELECT \* FROM SESSIONS

How to Close/Kill currently running session?

Using "CLOSE\_SESSION" Or "INTERUPT\_STATEMENT" commands, we can close/kill current running sessions.  
  
**Detailed Description:**  
If we need to close a session, follow the below steps  
  
Step1: Identify the session

* Find the Session Id which we need to close. Connect to Vertica DB Using super user and run the below query, it gives the list of current running sessions along with Session\_Id.

          SELECT \* FROM  SESSIONS  
  
Step2: Close the session

* Using "Close\_Session" or "Interupt\_Statement" commands we can colse/kill the current running sessions.

Syntax:

          SELECT CLOSE\_SESSION('<Session ID>')  
          OR  
          SELECT INTERUPT\_STATEMENT('<Session ID>')

----------

User Creation

GRANT USAGE ON SCHEMA PUBLIC

Create Schema

Grant usage privileges on a schema to the user

Grant all privileges on a schema to the user

Set default search path

Change Vertica User Privileges from Existing User to New user

Execute Permissions on User Defined Functions For New Users

Grant Execute Permissions on ODBC

----------

--Vertica Table

**Creation**

In two ways we can create a table in vertica,

* Without projections
* With Projections

**Without Projection:**  
      If we don't mention projections while the time of table creation, default super projection will create (During data insertion) based on inserted data.  
  
Syntax:

CREATE TABLE <Schema Name>.<Table Name>

( <Column name0> <data type0>,

<Column name1> <data type1>,

<Column name2> <data type2>,

.

.

.

)

Example:

CREATE TABLE SAMPLE.SAMPLE1 (ID INT,NAME VARCHAR(50))

**With Projection:**  
      If we mention projection while the time of table creation, based on that projection will create along with table.  
  
Syntax:

CREATE TABLE <Schema Name>.<Table Name>

( <Column name0> <data type0>,

<Column name1> <data type1>,

<Column name2> <data type2>,

.

.

.

)

ORDER BY <Column Name1>,<Column Name2>,....

SEGMENTED BY HASH(<Column Name1>,<Column Name2>,...

ALL NODES;

Example:

CREATE TABLE SAMPLE.SAMPLE1 (ID INT,NAME VARCHAR(50)) ORDER BY ID SEGMENTED BY HASH(ID,NAME)ALL NODES;

Note: In order by section we need to mention key columns (frequently used columns),  
SEGMENTED BY is used for big tables. UNSEGMENTED BY is used for small tables.

Rename Table, Table Column Or View in Vertica

        Below syntax's to rename a Table, column and view.  
  
**Rename a Table:**  
               ALTER TABLE <Schema Name>.<Table Name> RENAME TO <New Name for that table>  
  
     Example:  
               ALTER TABLE Test.Sample1 RENAME TO SAMPLE2  
  
  
**Rename a Column:**  
                 ALTER TABLE <Schema Name>.<Table Name> RENAME COLUMN <Column Name>  TO  <New Name For that column>  
  
       Example:  
                ALTER TABLE Test.Sample RENAME COLUMN Sample\_Id TO Sample\_2;  
  
  
**Rename a View:**  
                ALTER VIEW <Schema Name>.<Table Name> RENAME TO <New Name>  
  
        **Example:**  
                 ALTER VIEW TEST.SAMPLE4 RENAME TO SAMPLE6

**Projection Creation**

   We can create projection on a table to improve the query performance.  
  
**Syntax:**  
   CREATE PROJECTION <Schema Name>.<Projection Name>  
   ( <Column Name1>, <Column Name2>,....)   --*table related all columns*  
   AS  
   SELECT <Column Name1>,<Column Name2>,...  --*table related all columns*  
   ORDER BY <Column Name1>,<Column Name2>,...  --*required key columns*  
   SEGMENTED BY HASH(<Column Name1>,<Column Name2>,.. )  --*required columns*  
   ALL NODES;  
  
**Example:**  
**Table Creation:**  
   CREATE TABLE SAMPLE.SAMPLE1 (ID INT,NAME VARCHAR(50))  
  
**Projection Creation:**  
   CREATE PROJECTION Sample.Proj\_sample  
   (id, name)  
   AS  
   SELECT id, name  
   FROM Sample.sample1  
   ORDER BY id  
   SEGMENTED BY hash(id,name) ALL NODES;

**Create A New Table Using Existing Table**

  Using below command we can create a new table using existing table.  
Syntax:  
CREATE TABLE <New Table Name> AS SELECT \* FROM <Existing table name>  
  
Example:  
CREATE TABLE SAMPLE\_TBL\_BKP AS SELECT \* FROM SAMPLE\_TBL  
  
If you need only table structure then use below command:  
CREATE TABLE SAMPLE\_TBL\_BKP AS SELECT \* FROM SAMPLE\_TBL WHERE 1 = 2

Create Table Partitions

Change the Owner of Table

Drop Partition

--Sequence:

Create Sequence Number

Using sequence number we can generate sequence numbers.  
  
Syntax:  
CREATE SEQUENCE <Sequence Name> INCREMENT BY <Increment Number> START <Starting Number> NO CACHE

Alter Sequence

Drop Sequence

Add Sequence Or Set Default value to a table

Drop Default Or Sequence on a Table

Change the Owner of Sequence

--Vertica Sessions

Close Session in Vertica

Currently Running Sessions in Vertica