JCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.-6, Issue-5, May 2018 E-ISSN: 2347-2693

A Detailed Study On Features of Data Warehousing Database-Vertica

Jisha Mariam Jose

1*Dept. Of CSE, New Horizon College Of Engineering , Bangalore, India

*Corresponding Author: jmjisha@gmail.com

Available online at: www.ijcseonline.org

21/May/2018, Published: 31/May/2018

Abstract— The data which are to be stored and analyzed for various purposes have gone beyond the storage limit of the traditional relational database system. This has led in emerge of various big data technologies to store and process this huge collection of varieties of data. Vertica is an HP enterprise product, which is used in data warehouses to store and perform data analysis that are stored for decades. Vertica is not only used in data warehouses but also it can be integrated with Hadoop ecosystem for big data analysis. This paper basically describes the architecture, features, storage, various operations in Vertica analytics database that has made Vertica to be used for managing and analysis of large volumes of fast-growing data for achieving higher performance in query intensive applications and data warehouses.

Keywords—Column Orientation, Hybrid Store, Projections, Partitions, Tuple Mover, High Availability, Automatic Database Designer

I. INTRODUCTION

Vertica is a big data database product from HP. It is also a data warehousing database for handling terabytes/petabytes/exabyte of data. Vertica is a massively parallel SQL RDBMS (Structured Query Language Relational Database Management System) that commercializes the ideas of the Column-Store Project [1]. And hence supports a subset of ANSI SQL-99 standard. Its provided with JDBC/ODBC (Java Database Connectivity/Open Database Connectivity) drivers and a command line client (vsql: an interactive utility of Vertica to type SQL commands)[2]. It basically runs on major Linux distributions (RHEL, Suse, Debian, and Ubuntu). Also, Amazon AMI (Amazon Machine Image) is available for running Vertica in the cloud. Vertica's so-called"Community Edition" mode supports up to 1 TB of data and a cluster of 3 nodes without the license [2]. For larger setups, it requires a license from HP [3].

The main aim of this system is to achieve the features that users always expect from commercial RDBMSs, such as ACID transactions, a standardized declarative query language, security, high availability, etc., but with an architecture that is more focused on optimization of analytical queries for higher performance rather than transaction processing [4]. As the table size is growing for smaller companies also in much faster rate, the analytic workload has also increased. By concentrating on analytic workloads, it is possible to improve the performance in orders of magnitudes for existing one-size-fits-all systems[5].

In this paper, the sections are organized as follows: Section I contains introduction on Vertica Database, Section II contains Vertica object hierarchy that describes how logical description of tables are converted to containers that are physically stored in Vertica database, Section III contains detailed description on hybrid storage model of Vertica that deals with different types of data load and how data movement happen from primary memory to disk, Section IV describes the complete study on features of Vertica with examples which led Vertica to be used in big data analytics, Section V explains various types of projections and how they are formed, Section VI describes Vertica's partitioning concept with example and Section VII includes the conclusion.

II. VERTICA OBJECT HIERARCHY

It is helpful to understand the following terms when using Vertica [3]:

Host: A computer system with RAM(Random Access Memory), hard disk, Intel or AMD processor of 32-bit or 64-bit and TCP/IP(Transmission Control Protocol/Internet Protocol) network interface (IP address and hostname). Hosts share neither disk space nor main memory with each other [3].

Instance: An instance of Vertica consists of a Vertica process in running state and disk storage to store catalog and

© 2018, IJCSE All Rights Reserved

data on a host. Only one instance of Vertica can be in running state on a host at any particular time [3].

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

Cluster: A collection of hosts (nodes) bound to a database. A cluster is not part of a database definition and does not have a name [3].

Database: A cluster of nodes that can perform data storage in a distributed manner and execute SQL statement on active mode through administrative, interactive, and programmatic user interfaces [3].

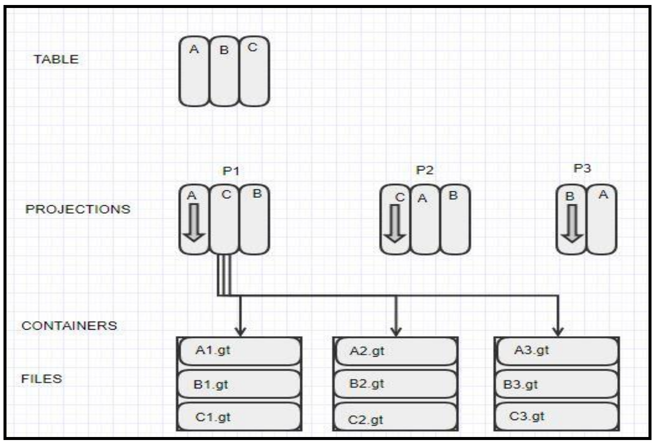


Figure 1. Vertica Object Hierarchy

Table: Vertica supports tables as a logical concept called as anchor table. A table has one or more projections in Vertica.

Projection: A projection is a collection of table columns. They are physical storage for table data. Data in a projection is compressed, encoded and sorted making it optimized for query execution. In the above figure 1, downward arrow marks indicate the columns on which a particular projection is sorted. The two basic types of projections in Vertica are:

(1) <u>Super projection:</u> It contains every column of a table.
(2) <u>Query-specific projection</u>: It contains only the subset of table columns.

For every table, Vertica creates at least one super projection. At query execution, the optimizer chooses the best projection for the query. We can create projections using the Vertica Database Designer or create them manually. Vertica automatically creates projections on first data load into a table. Data is loaded through tables into projections. We can also manually create projections for existing tables using the CREATE PROJECTION statement as follows:

CREATE PROJECTION projection name (projection_col,...)
AS SELECT table_col,...FROM existing_table ...

Container: Vertica stores the projection data in ROS containers. Logical grouping of all data files (data file + metadata file) on a node against a projection is called a container. There can be more than one container for a single projection on a particular node. This happens because every time on insert or delete operation new data files (or broadly, containers) are created and not appended to the older files. But this limitation is only for 10 min because after 10 min (default time, can be configured also) tuple mover will perform merge out action to merge all the containers of a single projection into a bigger one. We can have 1024 containers per projection per node.

III. HYBRID DATA STORE

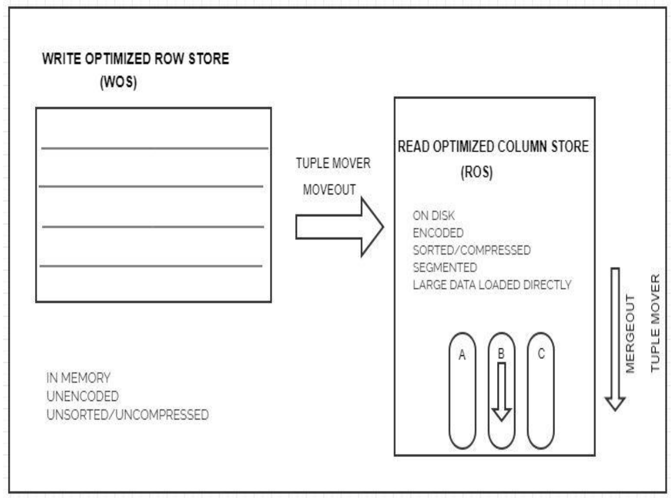


Figure 2. Hybrid Store- WOS & ROS

Vertica is unique in many ways, one of which can be seen in its data storage model. To understand the Vertica storage model, we first need to understand these three elements:
ROS (Read-Optimized Store)
WOS (Write-Optimized Store)
Tuple Mover
Vertica uses two distinct structures for storing data: WOS and ROS.

WOS – Write Optimized Store:
In Vertica, WOS means Write Optimized Store. The WOS is used for low-latency data loading and data storage is done in-memory [3]. It is designed to support INSERT, UPDATE, DELETE, and COPY operations. Initially, some small data are loaded to memory (WOS) in row format. Data stored in WOS for some time are also allowed to query. But they are unoptimized that means without compression or indexing to

© 2018, IJCSE All Rights Reserved

support faster loading and volatile. It will take the longest time to return results for data in WOS.

ROS – Read Optimized Store:

ROS, on the other hand, is structured for fast reads. ROS means Read Optimized Store. This disk storage structure is read oriented and highly optimized. ROS data are partitioned into separate sections known as storage containers. A container is just a set of column-wise data files created by moveout or COPY DIRECT statements and stored in a particular group of files. These data are both sorted and compressed that are stored in ROS of a database.

Vertica stores two files per column within a ROS container [6]: one with the actual column data, and one with a position index. Data are identified by a position that means ordinal position within a file in each ROS container. Positions are implicit and are never stored explicitly [6]. The position index is approximately (1/ 1000) the size of the raw column data and stores metadata per disk blocks such as start position, minimum value and maximum value that improve the speed of the execution engine and permits fast tuple reconstruction.

Tuple Mover:

Tuple Mover of Vertica is used to move data from WOS to ROS. The Tuple Mover performs two operations: Moveout and Mergeout.

Moveout:

During moveout operations, the Tuple Mover writes data to temporary space, where they are stored in columnar, sorted, encoded format. Once data re-organization is complete, moveout task moves data to ROS, creating new ROS containers for the new data. ROS containers are created and data are organized into projections on the disk. When data are committed to disk, data are removed from temporary space. Since the new data coming to ROS after moveout operations need not be merged with existing ROS data immediately, the movement of data in ROS container is faster. The tuple mover moveout task is performed whenever WOS reaches maximum capacity or by default after every 5min.

Merge out:

During the Tuple Mover's mergeout task, the small ROS containers that were created during moveout operations or on COPY DIRECT statements are combined or merged into larger containers. It is important because query from different containers will lead to slow results. Mergeout task compresses multiple containers to fewer containers. This allows the query to run more efficiently. It also purges data that is marked for deletion. By default mergeout task run automatically every 10 min.

Need For the Hybrid Model

Vertica uses the hybrid model to support different types of load. We can get data into Vertica database using any of these three methods:

(1) Load data into WOS and let the Tuple Mover move it to ROS (Auto):

Vertica loads data into WOS and continues loading directly to ROS if WOS becomes full. Size of WOS is 25% or 2GB of available RAM.

(2) Load data directly into ROS using the DIRECT option (Direct):

If data loaded to WOS exceed the size, the data are automatically ***spill over*** to ROS. That means, for Bulk load or large load: the best practice is to load data directly to ROS.

(3) Trickle load data only into WOS (Trickle):

Vertica loads small amount of data incrementally. This method is used only when we are confident that the WOS can hold the data, we are loading. Whenever the WOS becomes full, the entire data load is rolled back and error occurs.

When we use the COPY, INSERT, or UPDATE statements, data are loaded into WOS first by default and then after a specified interval of time data are loaded to ROS. We can manually load data directly into ROS also using the COPY DIRECT, INSERT DIRECT, or UPDATE DIRECT options.

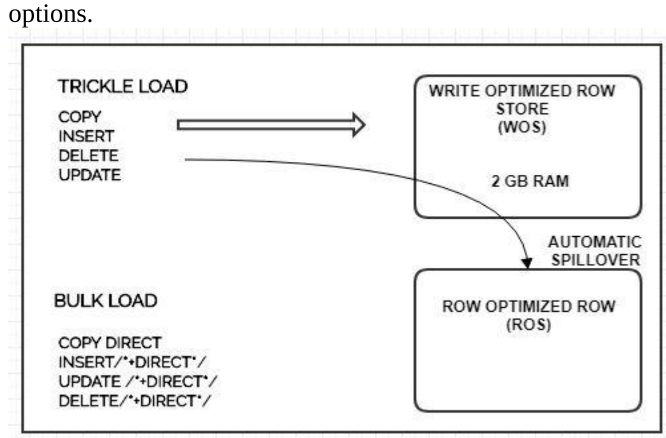


Figure 3. WOS –ROS With Different Load

IV. FEATURES OF VERTICA ANALYTICS PLATFORM

There major features which make it different from traditional RDBMS are:

Columnar Orientation, Advances Compression, High Availability, massively parallel processing, application Integration, Automatic Database Design.

A. COLUMNAR ORIENTATION

Vertica stores data in a column format so it can be queried for best performance. Each column of the table is stored separately as a data file on the disk. It is ideal for workloads

© 2018, IJCSE All Rights Reserved

which are more read-intensive. VERTICA reads only those column's data files which are needed to answer the query. Due to this, column storage reduces disk I/O (Input/output) operations when compared to row-based storage.

For example: The following figure 4 contains logical anchor table that has some information about a shop.

SHOP TABLE

CUST_ID	PURCHASE DATE	PRICE	PRODUCT_ID	QUANTITY
101	2017-12-21	100	P1	5
102	2017-12-21	20	P1	1
103	2017-12-21	146	P2	1
104	2017-12-15	45	P3	1
105	2017-12-15	1000	P5	1
101	2017-12-15	2000	P4	4
104	2017-12-15	135	P3	2

Figure 4. An Anchor Table

In typical RDBMS, the above table is physically stored as Row-Wise. So, in such system the following query is executed as follows:

QUERY: SELECT SUM(price) FROM shop WHERE cust_id=101;

First, all rows are read to search customer with id **101** and then those rows are filtered out. Then, on these selected rows, price column is scanned to find the sum.

ROW-WISE STORAGE

101	2017-12-21	100	P1	5
102	2017-12-21	20	P1	1
103	2017-12-21	146	P2	1
104	2017-12-15	45	P3	1
105	2017-12-15	1000	P5	1
101	2017-12-15	2000	P4	4
104	2017-12-15	135	P3	2

Figure 5. Row-Wise Storage in RDBMS

But in case of Vertica database, the figure 4, anchor table is physically stored as Column-wise that means each column is stored as separate data file as shown in below figure.

COLUMN-WISE STORAGE

CUST_ID	PURCHASE	PRICE	PRODUCT ID	QUANTITY
101	2017-12-21	100	P1	5
102	2017-12-21	20	P1	1
103	2017-12-21	146	P2	1
104	2017-12-15	45	P3	1
105	2017-12-15	1000	P5	1
101	2017-12-15	2000	P4	4
104	2017-12-15	135	P3	2

Figure 6. Column-Wise Storage in Vertica Database

Here, the ***QUERY: SELECT SUM(price) FROM shop WHERE cust_id=101;*** is executed as follows:

FIRST, only two column data files (cust_id, price files) are read and left all data files are ignored. And then cust_id 101 and its corresponding prices are filtered out as the solution for the above query. The information about which are the values corresponding to customer id 101 in price data file is obtained with the help of meta-data files which are stored along with every data file in ROS container.

B. ADVANCED COMPRESSION

Vertica uses encoding and compression to optimize query performance and save storage space.

Encoding: Encoding converts data into a standard format called as encoded data, which increases performance because there is less disk I/O during query execution[3]. Vertica uses a number of different encoding strategies, depending on column data types: table cardinality, and sort order. In addition, it can store more data in less space. It also passes encoded values to other operations, saving memory bandwidth. Vertica can directly process encoded data because Verica query optimizer can directly understand the encoded data which is encoded using Vertica's encoding strategies. The cardinality ratio means the dataset with more distinct values will have higher cardinality ratio.

Encoding Types

1. ***Auto****:* The system goes through properties of data and picks up the most appropriate encoding type automatically. It is the default type and is mostly used when insufficient usage examples are known [6].

2. *RLE**:* It means Run Length Encoding. It replaces sequences of identical values with a single pair that contains the value and number of occurrences. This type is best for sorted columns with low cardinality values [6],[7].

3. *Delta Value**:* Data is recorded as a difference from the smallest value in a data block. This type is best for unsorted columns with high cardinality.

4. *Block Dictionary**:* Here, within a data block, distinct column values are stored in a dictionary and actual values are

© 2018, IJCSE All Rights Reserved

replaced with references to the dictionary. This type is best for few-valued, unsorted columns such as stock prices.

5. ***Compressed Delta Range*****:** It stores each value as a delta from the previous one. This type is ideal for many-valued float columns that are either sorted or confined to a range.

6. ***Compressed Common Delta***: It builds a dictionary of all the deltas in the block and then stores indexes into the dictionary using entropy coding. This type is best for sorted data with predictable sequences and occasional sequence breaks. For example, timestamps recorded at periodic intervals or primary keys.

Compression: Compression transforms data into a compact format. Vertica uses integer packing for unencoded integers and LZO(Lempel–Ziv–Oberhumer) for compressed data. Using compression, Vertica stores more data, and uses less hardware than other databases. Using compression, it is possible to store much more historical data in physical storage. Before Vertica can process compressed data it must be decompressed.

Compression allows a column store to occupy substantially less storage than a row store [3]. In a column store, every value stored in a column of a projection has the same data type. This greatly facilitates compression and encoding techniques, particularly in sorted columns. In a row store, each value of a row can have a different data type, resulting in a much less effective use of compression.

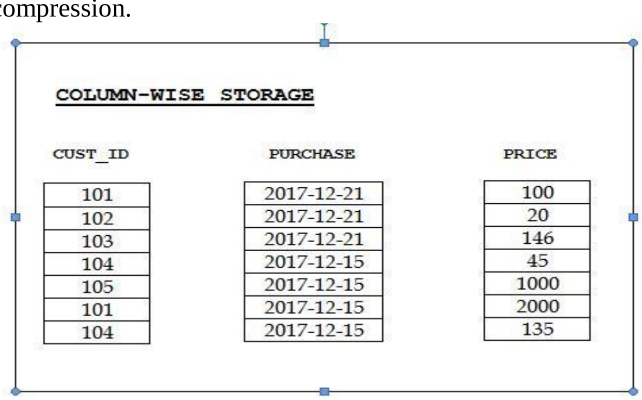


Figure 7. Data Files Before Encoding & Compression

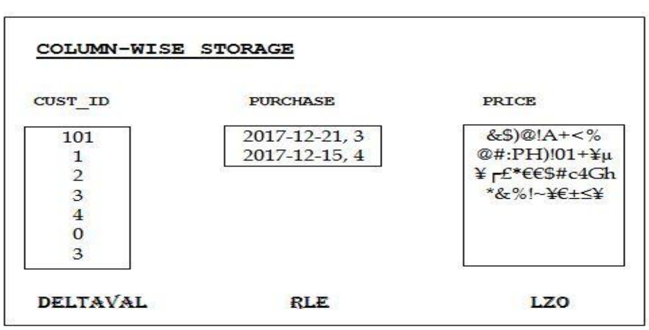


Figure 8. Data Files in Vertica After Encoding & Compression

Let's see in this case how a query is executed from encoded and compressed files[8].

QUERY: **SELECT cust_id FROM shop WHERE purchase='2017-12-15';**

Because of column-oriented storage, only two data files **cust_id, purchase** are scanned. When **"purchase"** data file is referred in above figure 8, the values **2017-12-21, 3** indicates :

First 3 records do not match with query's filter value **"2017-12-15"**, so ignore the corresponding first 3 records in cust_id data file also. This connectivity between data files are identified with the help of metadata files stored along with each data file in ROS container.

C. HIGH AVAILABILITY

Vertica provides high availability of the database by making multiple copies of the same data on different nodes. Even if the node is down in Vertica, the loading and querying of data are performed. In Vertica, the node after its recovery, automatically queries the other nodes to recover missing data. Clustering supports scaling and redundancy. Database cluster can be scaled by adding more hardware. Reliability can be improved by distributing and replicating data across the cluster.

K-safety sets the fault tolerance in the database cluster. The value K also represents the number of copies of segmented data in the database cluster. In Vertica, the value of K can be zero (0), one (1), or two (2). It means, even if any one node in a database with a K-safety value of one (K=1) goes down, the database still continues to run normally and all the data will be available. When a node is added to the cluster or comes back online after being unavailable, it automatically queries other nodes to update its local data [3].

If more than the value of K, the number of nodes get fail, some of the data in the database will become unavailable. In this case, the database is considered as unsafe and automatically shuts down. However, if every data segment is available on at least one functioning node, then Vertica continues to run safely.

Buddy Projections

Vertica creates copies of segmented projections of same data that are distributed across nodes in a cluster that also determines the value of K-safety. These copies are called as buddy projections. This ensures that even if the nodes go down according to K-safety value, all the data will be still available from the remaining nodes.

The way Vertica stores these copies is that it divides each table into n number of pieces, n being the number of nodes. then it copies each piece into each node starting with say node 1. it then copies the pieces from the second copy on

© 2018, IJCSE All Rights Reserved

each node again, but this time starting at node 2. Similarly, it copies the pieces of the third copy starting with node 3.

Example 1: In the below 3-node cluster figure 9, ksafety- 1 is achieved with one buddy projection in each neighbouring nodes. If node-2 goes down, still database is safe because segment A can be obtained from node-1, segment B from node-3's buddy projection and segment C from node-3.

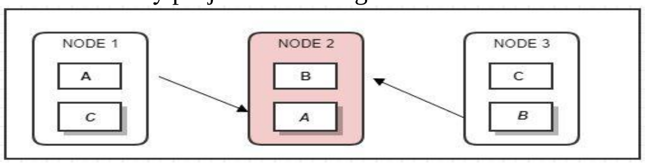


Figure 9. Three Node Cluster with one Buddy Projection

Example 2: In the below figure 10, if any one node goes down (say node 2), still all the segments are available for the database to be in the safe state. So, for the below five node cluster, high availability with ksafety -1 is always achievable. Let's see, if ksafety-2 is achievable for the same orientation of nodes.

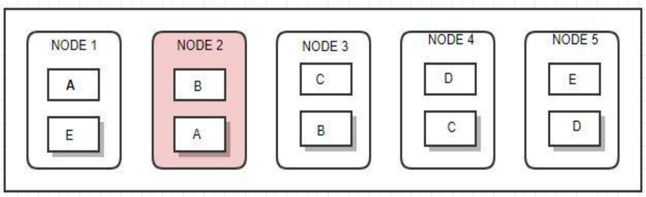


Figure 10. Five Node Cluster with one Buddy projection

Example 3: From the figure 11 & figure 12, if two nodes- node 2 and 4 goes down still database will be in the safe state as all the segments are available from remaining nodes. But if node 1 and 2 goes down, the database will be unsafe as all segments (segment A) will not be available. So, for five nodes cluster with k-safety 2 is not always high available for the below orientation.

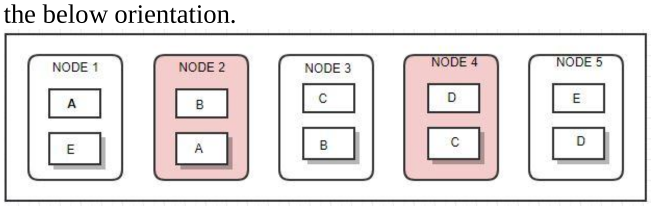


Figure 11. Five Node Cluster With Node-2, Node-4 As Down

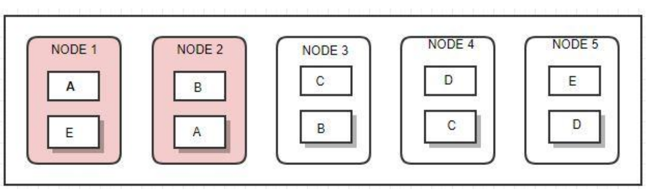


Figure 12. Five Node Cluster With Node-1, Node-2 As Down

Example 4: A shown in the below figure 13, for five node cluster to achieve high availability with ksafety-2, we must have minimum two buddy projection for every node. Here, if any two nodes go down, still all the segments are available.

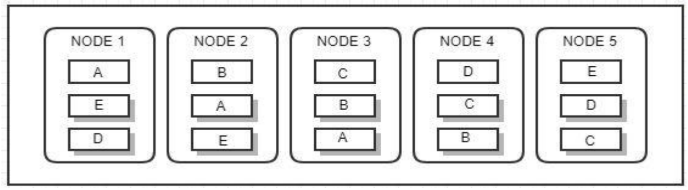


Figure 13. Five Nodes Cluster with Two Buddy Projections

<u>Examples of some cases where the database will function properly even if half of the nodes in a cluster are down.</u>

K-safety = 1 means two things :

i) ANY one node can fail in the cluster.

ii)One less than half the number of nodes can fail in the cluster as long as no two nodes are adjacent to each other. So if you have 10 nodes then the database can be running even if 4 nodes go down.

Example 5: The above fig 14, shows a 10 node cluster with a k-safety of 1. Node 2, 4, 6 and 8 have failed, yet the cluster will continue functioning as all the segments are available from remaining nodes. So with a k-safety of 1, up to one less than 50% of the nodes i.e. 4 nodes can go down.

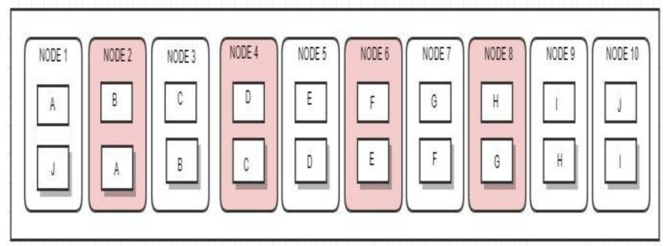


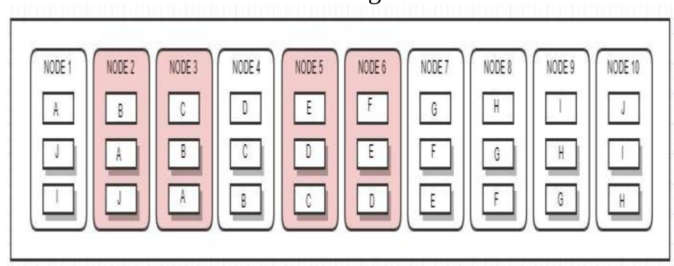
Figure 14. Ten Nodes Cluster With One Buddy Projection, Node 2,4,6,8 are down

k-safety=2 means :

i)ANY two nodes can fail in the cluster.

ii)As long as not more than three contiguous nodes fail, the cluster can take the failure of the other nodes with the limit of 1 less than 50% of the nodes.

Example 6: The above fig 15, shows a 10 node cluster with a k-safety of 2. Node 2, 3, 5 and 6 have failed, yet the cluster will continue functioning as all the segments are available from remaining nodes.



© 2018, IJCSE All Rights Reserved

Figure 15. Ten Nodes Cluster With One Buddy Projection, Node 2, 3, 5, 6 are down

Since the probability of more than two continuous nodes failing even in large clusters is pretty low. This enables us to create resilient clusters with tens or a couple hundred nodes.

D. AUTOMATIC DATABASE DESIGN

Vertica's Database Designer(DBD) is a tool that is used for analysing the logical schema, sample data, and sample queries as an option. It recommends projection design that provides the best performance for the user's workload. It is also used to create a data storage and projections design that can be deployed automatically or manually that can be used by anyone without specialized database knowledge that is even business users can run Database Designer. We can run and re-run the DBD(Database Designer) any number of times for additional optimization without stopping the database[3]. One can store various projections with different column sort orders of a particular table. By this, on run of DBD, it automatically selects the best projection for the given set of queries[7].

Every projection has its own set of data file and projections doesn't share their data file. Every column has a separate data file, where data get stored against projection in vertica. On creation of table, projection is not formed. But on a load of data for the first time, a projection is created [may not be optimized]. On update or deletion of further data, projections are also updated accordingly.

Explanation:
Suppose on initial load of data projection p1 is created for table T1[9]. In Vertica, when data is loaded for the first time: a super projection and buddy projection [according to k-safety value] is created. Here, these projections may not be optimized very well because on load time only a few information is known. Such projections are called as **unoptimized super projection**. Then for query Q1, on run of automatic database design following thing happens:

DBD analyses data and query and if required prepare a separate projection(say p2) for faster retrieval of query Q1. For next query Q2, if DBD is run again: if previous projections are sufficient for query Q2, then no additional projection is created. But if it analyses data, Q2 and finds better projection can be created, then P3 is created and so on.

Query specific projection is the projection formed on running of database designer for a specific query and during which it is analyzed that a new projection with a subset of columns of a table has to be created for faster performance**.** Initial or original projection, which has been created on all columns of a table during the initial load of data is called **super projection.**

Projection can be created manually also by using manual creation syntax. We can use the Database Designer to create one of the following types of designs:

- **A comprehensive design** that allows creating new projections for all tables in the database in one stretch.
- **An incremental design** that creates projections for all tables that are referenced in the queries that are supplied to DBD.

Comprehensive mode:
A comprehensive design creates an initial projection when DBD is run for the first time or replacement design for all the tables in the specified schemas of the database. When we run DBD for the first time it is recommended to run in comprehensive mode. Basically, it replaces all unoptimized projections with optimized projections.

To help Database Designer in creating an efficient design, the representative data has to be loaded into the tables before the beginning of the design process. When you load data into a table, Vertica creates an unoptimized super projection so that Database Designer has projections to optimize. If a table has no data, Database Designer cannot optimize it [3].

Optionally, supply Database Designer with representative queries that are planned, so that Database Designer can optimize the design for them[10]. If queries are not supplied, Database Designer creates an optimized super projection in general that minimizes storage, but query-specific projections are not created in such case.

Incremental mode:
After you create and deploy a comprehensive database design, it's likely that your database will change over time in various ways. Database Designer is run in incremental design mode to address these changes in database schema [3].

- Significant data additions or updates
- New or modified queries that you run regularly
- Performance issues with one or more queries
- Schema changes

It will not replace existing unoptimized or optimized super projection as it might be used by some other queries. It only creates a new one, only if required. Design queries are required for incremental designs.

Database Designer yields the following output[3]:

- A design script that creates the projections for the design in a way that meets the optimization objectives and distributes data uniformly across the cluster.
- A deployment script that creates and refreshes the projections for your design. For comprehensive designs, the deployment script contains commands that remove non-optimized projections. The deployment script includes the full design script.

© 2018, IJCSE All Rights Reserved

- A backup script that contains SQL statements to deploy the design that existed on the system before deployment. This file is useful in case you need to revert to the pre-deployment design.

Optimization Objectives

Optimize for query performance

- Generate a set of candidate projection for each query.
- Invokes the optimizer to determine query costs of projections and pick the one with the lowest cost

Optimize storage footprints

- Tries every possible encoding and compression type on every column.
- For each column, select the encoding and compression type, that most reduces the size

Balanced design

- Database Designer creates a design whose objectives are balanced between database size and query performance.

A fully optimized query has an optimization ratio of 0.99.

E. MASSIVELY PARALLEL PROCESSING(MPP)

Vertica is a shared architecture. It allows each node in the cluster to work on its portion of a database when running a query. The public network is used for communication with outside world and the Private network is used for intra node communication(query plans, query results, data loads). All nodes in the cluster are peers. We can load data continuously in real time to any node. The request will be equally distributed and managed by making one of the node initiators of query execution and others as executors.

In other words, SQL query is written against tables. In order to execute a query, the Vertica database generates a query plan. A query plan is the sequence of steps used to determine the execution path and resource cost for each step. The cost calculated at each step in a query plan is the estimation of resources used like:-

- Data distribution statistics
- Disk space
- Network bandwidth
- CPU speed
- Data segmentation across the cluster

When you submit a query, the initiator chooses the projections to use, optimizes and plans the query execution. Planning and optimization are quick, requiring at most a few milliseconds.

The query plan that the optimizer produces after choosing one of the projections is further broken down into "mini-plans." These mini-plans are distributed to the other nodes, known as executors. The nodes process the mini-plans in parallel, interspersed with data movement operations. The query execution proceeds with intermediate result sets (rows) flowing through network connections between the nodes as needed.

In the final stages of executing a query plan, some wrap-up work is done at the initiator, such as:-

- Combining outcomes after the group by operation
- Merging all the partial outputs received from all the executors
- Formatting these aggregated results to return to the format which is understood by the client.

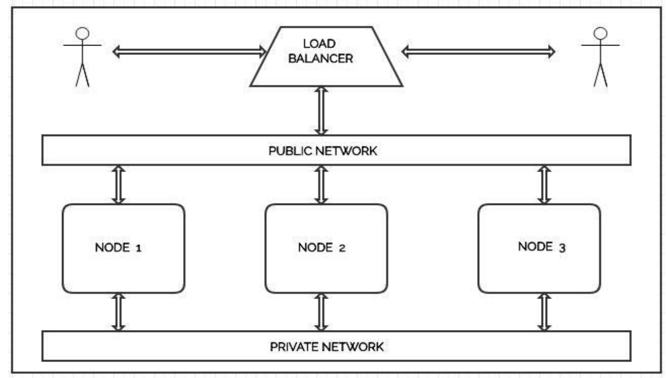


Figure 16. Broader View of 3 Node Cluster in the Network

F. APPLICATION INTEGRATION

Vertica is integrated with various technologies that include tools starting from ETL(Extract Transform Load) tools, data store management , analyzing the results, Business intelligence, and visualizing as dashboard etc. Vertica supports industry-standard drivers, such as ODBC and JDBC, for connecting client applications to Vertica.

Vertica has native integration with open source big data technologies like Apache Kafka and Apache Spark. It supports for standard programming interfaces, including ODBC, JDBC, ADO.NET(ActiveX Data Objects (**ADO**) technology), and OLEDB(Object Linking and Embedding, Database). It also supports high-performance and parallel data transfer to statistical tools such as built-in machine learning algorithms based on R-language, and the ability to store machine learning models, and use them for in-database scoring.

V. VERTICA PROJECTIONS

A. TYPES OF PROJECTIONS

1. Super Projection
2. Query-specific Projection
3. Buddy Projection
4. Pre-Join Projection
5. Live Aggregate Projection

(1) Super Projection:

A *super projection* contains all the columns of a table. For each table in the database, Vertica requires a minimum of

© 2018, IJCSE All Rights Reserved

one projection, which is the super projection. Super projection is the one that Vertica automatically creates when we initially load data into a table using INSERT, COPY commands. A table can have multiple super projections.

(2) *Query-specific Projection:*
A *query-specific projection* is a collection of a subset of columns of a table that is used to process a specified given query. Query-specific projections significantly improve the performance of those queries for which they are optimized.

(3) *Buddy Projection:*
A projection with the same columns and segmentation on different nodes to provide high availability.

(4) *Pre-join projection*:
In pre-join projection, multiple tables are joined and stored in the form of projection. These are manually created. A *pre-join projection* contains inner joins between tables that are connected by primary key or foreign key constraints. Pre-join projections provide a significant performance advantage over joining tables at query run time.

(5) *Live Aggregate Projection:*
These are manually created projections that contain aggregated data. A live aggregate projection contains columns with values that are aggregated from columns in its anchor table. When we load data into the table, Vertica aggregates the data before loading it into the live aggregate projection. On subsequent loads—for example, through INSERT and COPY—Vertica recalculates aggregations with the new data and updates the projection. Since aggregated data is already present in these projections, the queries with aggregate functions such as SUM, COUNT, MIN, MAX etc. are executed more efficiently.

B. VERTICA DISTRIBUTION OF DATA ON DIFFERENT NODES.

There are two methods of distribution:
(1) ***Replication**:* It is the process of copying the full projection to each node. This method is used for small projections such as projections with less than 1million records.
(2) ***Segmentation:*** It is the process of segmenting and distributing the projection data across multiple nodes. This method is used for large projections.

Vertica creates copies of segmented projections that are distributed across database nodes known as buddy projections.

Segmented Projections :
We typically create segmented projections for large tables. Vertica splits segmented projections into chunks (segments) of similar size and distributes these segments evenly across the cluster. K-safety determines how many duplicates (*buddies*) of each segment are created and maintained on different nodes. We create segmented projections with a CREATE PROJECTION statement that includes a SEGMENTED BY clause[3].
Projection segmentation achieves the following goals:
- Ensures high availability and recovery.
- Spreads the query execution workload across multiple nodes.
- Performs optimization of each node according to the given query workloads

Vertica uses hash segmentation to segment large projections. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution.

In a projection, the data to be hashed can be one or combination of columns that have a large number of unique values. For this, usually primary key columns are used as hash function arguments.

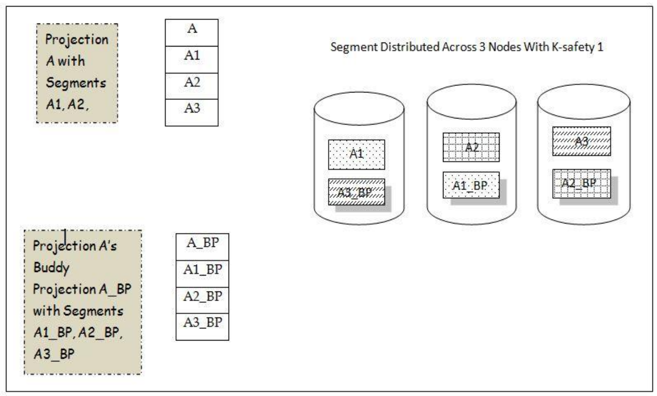


Figure 17. Segment Distribution Across 3 Nodes with K-safety 1

Unsegmented Projections(Replication):
HP Vertica performs replication in case of small, unsegmented projections and creates buddy projections in case of large, segmented projections for ensuring high availability and recovery for database clusters of three or more nodes.

When it creates projections, Database Designer does not segment projections for small tables; rather it replicates them, creating and storing duplicates of these projections on all nodes within the database.

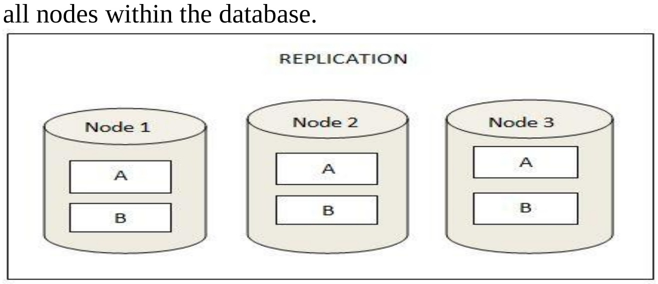


Figure 18: Replication Distribution Across 3 Node Cluster

© 2018, IJCSE All Rights Reserved

We can also manually create un segmented projections with a CREATE PROJECTION statement that includes the clause UNSEGMENTED ALL NODES. This clause specifies to create identical instances of the projection on all cluster nodes.

Creation of Projections Manually

The components of projection are:

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

For Example

The following is a table "hello" in schema "test" shown from VSQL editor of Vertica(3 node cluster)

*dbadmin=> select * from test.hello;*

eno	name	age	dno
1002	ramana	21	10
1003	rohit	22	30
1004	romy	23	20
1001	ram	21	10
1200	MEERA	25	20
1211	qwerty	23	30

CASE 1 : Creation of manual projection depicting Replication with ksafe = 1

dbadmin=> create projection hello_m1(eno encoding deltaval, age encoding RLE) as select eno, age from test.hello order by eno ksafe 1;

where,

Column List and Encoding : (eno encoding deltaval, age encoding RLE)

Base Query: select eno, age from test.hello

Sort Order: order by eno

Segmentation: No segmentation, that means its replication

dbadmin=> \dj

List of projections

Schema	Name	Owner	Node	Comment
test	hello_b0	dbadmin		
test	hello_b1	dbadmin		
test	**hello_m1**	**dbadmin**	**v_nhdb_node0002**	
test	**hello_m1**	**dbadmin**	**v_nhdb_node0001**	
test	**hello_m1**	**dbadmin**	**v_nhdb_node0003**	

Here ,a new projection "hello_m1" is replicated on all the 3 nodes of the cluster.

*dbadmin=> select * from test.hello_m1;*

eno	age
1001	21
1002	21
1003	22
1004	23
1200	25
1211	23

CASE 2: creation on manual projection depicting segmentation with ksafe = 1

dbadmin=> create projection hello_m2(eno encoding deltaval, dno encoding RLE) as select eno, dno from test.hello order by eno segmented by hash(eno) all nodes ksafe 1;

where,

Column List and Encoding : (eno encoding deltaval, dno encoding RLE)

Base Query: select eno, dno from test.hello

Sort Order: order by eno

Segmentation: segmented by hash(eno) all nodes

dbadmin=> \dj

List of projections

Schema	Name	Owner	Node	Comment
test	hello_b0	dbadmin		
test	hello_b1	dbadmin		
test	hello_m1	dbadmin	v_nhdb_node0002	
test	hello_m1	dbadmin	v_nhdb_node0001	
test	hello_m1	dbadmin	v_nhdb_node0003	
test	**hello_m2_b0**	**dbadmin**		
test	**hello_m2_b1**	**dbadmin**		

For segmentation with ksafe=1, one buddy projection is created. Since data is segmented across three nodes internally; the distribution of these segments of a particular projection on three nodes are not shown here.

*dbadmin=> select * from test.hello_m2_b0;*

eno	dno
1002	10
1003	30
1004	20
1001	10
1200	20
1211	30

CASE 3: Creation of manual projection depicting Replication with ksafe= 0

© 2018, IJCSE All Rights Reserved

dbadmin=> create projection hello_m5(eno encoding deltaval, age encoding RLE) as select eno, age from test.hello order by eno ksafe 0;

where,
Column List and Encoding : (eno encoding deltaval, age encoding RLE)
Base Query: select eno, age from test.hello
Sort Order: order by eno
Segmentation: No segmentation, that means replication

dbadmin=> \dj;

List of projections

Schema	***Name***	***Owner***	***Node***	***Comment***
test	***hello_m5_b0***	***dbadmin***		
test	***hello_m5_b1***	***dbadmin***		

Since, Replication and ksafe =0 (that means there is no availability) is a contradiction, Vertica system itself has taken the distribution as segmentation which is by default. And by default ksafe value for segmentation is 1. Hence one buddy projection is created.

CASE 4: Creation of manual projection depicting segmentation with ksafe =0

dbadmin=>create projection hello_m6(eno encoding deltaval, name) as select eno, name from test.hello order by eno segmented by hash(eno) all nodes ksafe 0;

where,
Column List and Encoding : (eno encoding deltaval, name)
Base Query: select eno, name from test.hello
Sort Order: order by eno
Segmentation: segmented by hash(eno) all nodes

dbadmin=> \dj;

List of projections

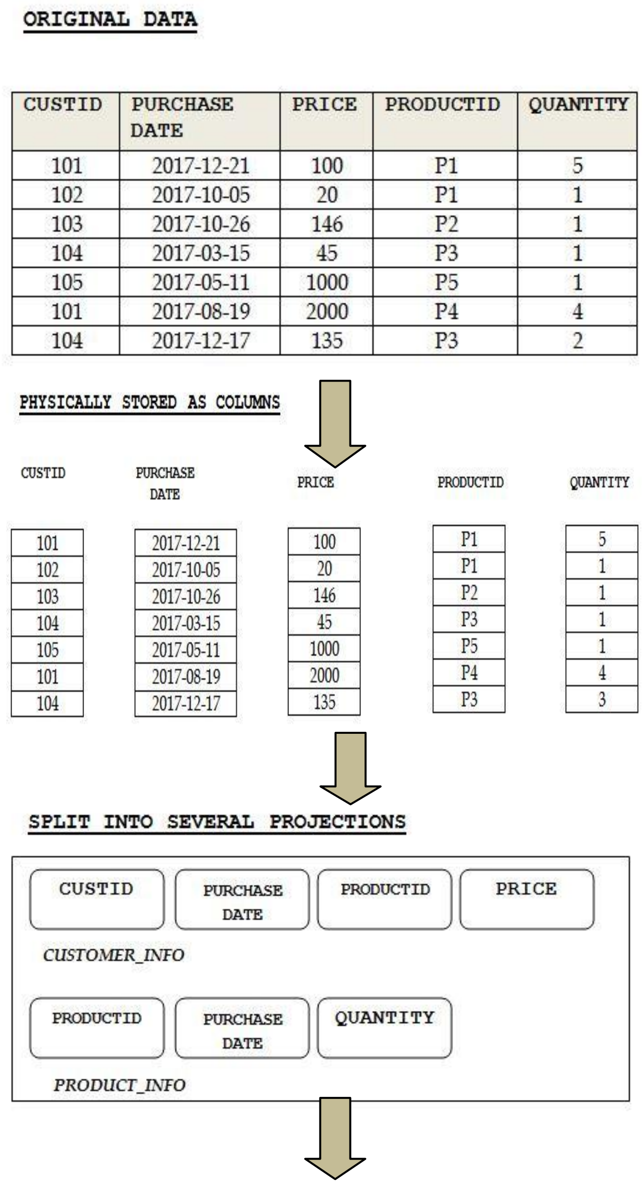
Schema	***Name***	***Owner***	***Node***	***Comment***
test	*hello_m5_b0*	*dbadmin*		
test	*hello_m5_b1*	*dbadmin*		
test	***hello_m6***	***dbadmin***		

Since ksafe=0, no buddy projection is created in this segmentation distribution.

VI. VERTICA'S PARTITIONING

HP Vertica supports data partitioning at the table level, that means partition is done for all the projections of a particular table which gets divided into smaller pieces. Partitions are commonly used when data in projections are divided based on time. For example, if we have data in the table over the years, then retrieval of information from such data can be made faster by partitioning the table by the year or by month if the data stored in the table are based on a particular year. To drop a partition, first data from different nodes are segregated and then a whole partition is dropped. Data management and performance of queries are improved due to partitions.

The below figure 19, shows an example of a logical table that is stored in two projections: cust_info, product_info. These projections are further segmented to machine1 and machine2. And last part of the figure shows how machine2's segmented data is further partitioned according to months[10].



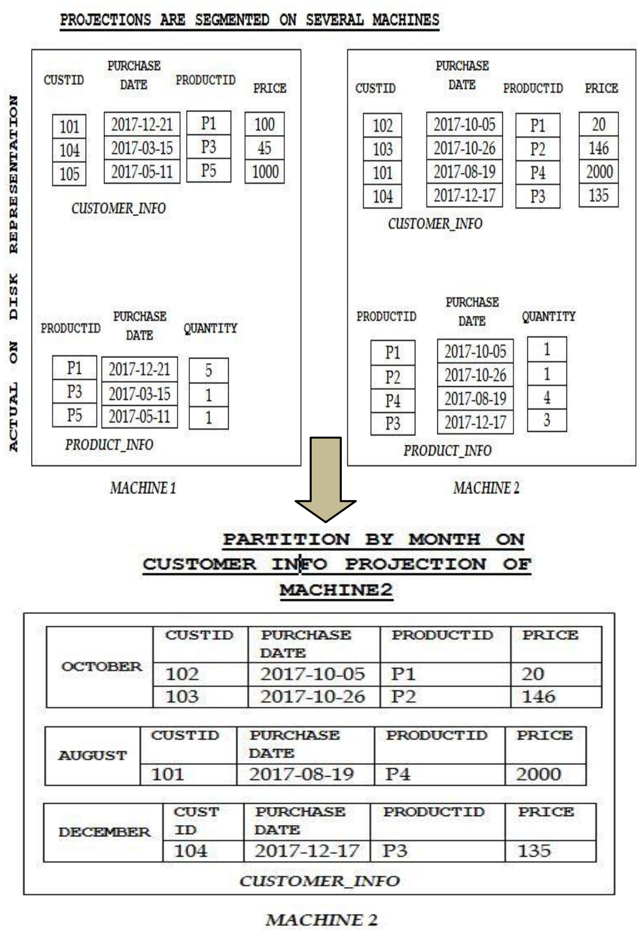


Figure 19. Storage of Data in hierarchical order: Anchor table, Projection, segments on different machine, Partition done on segments

The Vertica partitioning capability makes it easy to manage this data efficiently[10]. Now for the above figure 19, if a query is given as: SELECT *custid* FROM *customer_info* WHERE *purchase_date* LIKE ‘%-10-2017%; The following two level of pruning is done in vertica :

- Only selected column’s data file is fetched[that means custid, purchase_date] because of column-orientation property.
- Only a particular ROS container is searched [that means October container] because of partitioning.
- Hence, high performance is achieved in Vertica.

In above example partitioning is done for the year 2017 by “month” , so total 12 ROS container is created for the year 2017.

Case 1: When a new data is inserted for October-2017,

- Then a separate ROS container is created .
- Then on merge out, this container will merge with existing October-2017 container to become a larger container.

Case 2: When a new data is inserted for October-2018,

- Then a separate ROS container is created for October-2018.
- But this new container will not be merged with existing one because existing one is October-2017 and the new one is October -2018.

You can specify partitioning for a table when you initially define the table with *CREATE TABLE*. Also, *ALTER TABLE command* can be used to change the definition of an existing table. In the first case, whenever the data is inserted, it is automatically loaded to the respective partitions that are already created during the initial stage. Whereas in the second case, the existing data has to be repartitioned explicitly using Vertica function called *PARTITION_TABLE*. A table definition specifies partitioning through a *PARTITION BY* clause[3]:
PARTITION BY expression
Where**,** *expression* resolves to a value derived from one or more table columns. The column on which partitioning is done has to be not null constraint[3].

Example 1: Create partitioning on table “test” of schema ”nh014” month -wise
Create table nh014.test
(tsymbol char,
tdate date not null)
partition by extract (month from tdate);

Insertion of data is done as follows:
=>insert into nh014.test values(‘a’, ’01-Jan-2017’);
=>insert into nh014.test values(‘b’, ’12-Feb-2017’);
=>insert into nh014.test values(‘a’, ’23-Jun-2017’);
=>insert into nh014.test values(‘a’, ’28-Dec-2017’);
➔*commit;*

To verify partition formation using Vertica system table “partitions”
=>Select projection_name, ros_id, partition_key, node_name, ros_row_count from partitions where table_schema=‘nh014’ and projection_name=test_b0;

To do manually moveout :
Manually *TUPLE MOVER MOVEOUT* is done so that data which are in WOS after insertion are moved out to ROS instead of waiting for *MOVEOUT* action to happen automatically after some time.
⇒ *Select DO_TM_TASK(‘MOVEOUT’);*

To do manually mergeout
Manually *TUPLE MOVER MERGEOUT* is done so that data which are in newly created ROS container will be merged with already existing partition ROS container to become a single large container instead of waiting for *MERGEOUT* action to happen automatically after some time.
=>SELECT DO_TM_TASK(‘MERGEOUT’);

Example 2: Creation of table initially without partitioning

=> Create table nh014.test1
(tsymbol char,
tdate date not null);

Insertion of data is done as follows:
=>insert into nh014.test1 values('a', '01-jan-2017');
=>insert into nh014.test1 values('b', '12-Feb-2017');
=>insert into nh014.test1 values('a', '23-Jun-2017');
=>insert into nh014.test1 values('a', '28-dec-2017');
➔*commit;*

Addition of partition on the above table using ALTER TABLE syntax:

ALTER TABLE nh014.test1 PARTITION BY EXTRACT (MONTH FROM TDATE) REORGANIZE;

Example 3: Partitioning by Year and Month in combination

To partition by both year and month in combination, the partition clause should be like –append month indicated by two digits with year as an expression like this:
*PARTITION BY EXTRACT(year FROM tdate)*100 + EXTRACT(month FROM tdate)*
This expression formats partition keys as follows:
201701 201702 201803 ... 201711 201802

VII. CONCLUSION

As data is growing in faster rate in terms of volume, velocity, variety etc; it has become important to store, manage and perform analysis on such large data. This paper gives an overall idea on Vertica database's basic features and out of these, some features are common for other databases that are used in big data applications also. The column-store feature of Vertica helps in faster retrieval of queries from a huge collection of data whereas high availability describes how a database can be always available, even if some nodes are down according to k-safety value. Each section of this paper will give a deeper insight into every feature of Vertica, where some concept's practical implementation is also shown.

REFERENCES

[1] D.J. Abadi, P.A. Boncz, S.Harizopoulos, *"Column-oriented database systems"*, In The Proceedings of the VLDB Endowment, pp. 1664–1665, 2009.

[2] T. Siivola, *"A Short Introduction To Vertica"*, RedHat Software Developer Meetup, 2014.

[3] *"Vertica Analytics Platform",* Vertica Documentation, Version: 8.1.x, 2018.

[4] C. Bear, A. Lamb, N. Tran, *"The Vertica Database: SQL RDBMS For Managing Big Data",* In The Proceedings of the workshop on Management of big data systems, 2012.

[5] M. Stonebraker, *"One size fits all: an idea whose time has come and gone"*, In The Proceedings of 21st International Conference on Data Engineering, pp. 2-11, 2005.

[6] A. Lamb, et al., *"The Vertica Analytic Database : C-Store 7 Years Later"*, In The Proceedings of the VLDB Endowment vol.5, No.12, pp 1790–1801, 2012.

[7] D. Abadi, D. Myers, D. DeWitt, S. Madden, "Materialization Strategies in a Column-Oriented DBMS", In The Proceedings IEEE 23rd International Conference on Data Engineering, pp 466—475, 2007.

[8] S.Chakraborty , J. Doshi, *"Data Retrieval from Data Warehouse Using Materialized Query Database",* International Journal of Computer Sciences and Engineering, Vol.6, Issue.1, pp-280-284, 2018.

[9] Ramakrishna Varadarajan, V. Bharathan, A. Cary, J. Dave, S. Bodagala, *"DBDesigner: A Customizable Physical Design Tool for Vertica Analytic Database"*, In The Proceedings of IEEE 30th International Conference, pp. 1084-1095, 2014.

[10] "*A DBMS Architecture Optimized for Next-Generation Data Warehousing"* , The Vertica Analytic Database Technical Overview White Paper, Vertica System, 2010.

Authors Profile

Ms. Jisha Mariam Jose pursued Bachelor of Technology in Computer Science and Engineering from Govt. Engineering College, Thrissur, Kerala, India in 2009 and Master of Technology in Computer Information Science from Cochin University of Science and technology in year 2012 , after securing 97.53 percentile in GATE-2010. She is currently working as Assistant Professor in Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, India. Her main area of interests are: Big Data Analytics, Database Management System, Data Mining and Computer Networks. She has 3.5 years of teaching experience.

© 2018, IJCSE All Rights Reserved