ISSN 2278-7763

Column-Oriented Databases to Gain High Performance for Data Warehouse System

By

Nirmal Lodhi, PHD Research Scholar – Bhagwant University,
Email : - NirmalrbSingh@gmail.com
Ram Babu, PHD Research Scholar – Bhagwant University,
Email : - singh.rambabu@gmail.com
R.C. Verma, Asst. Professor IMT Faridabad,
Email: - rcvranchi5@yahoo.co.in
Rajesh Pandey, PHD Research Scholar – Bhagwant University,
Email : - rajesh.fbd@gmail.com

Abstract:-

Column-oriented database systems, also known as column-stores, have an important demand in the past few years. Basically, it is about storing each database column separately so that the attributes belonging to the same column would be stored contiguously, compressed and densely-packed in the disk. This method has advantages in reading the records faster as compared to classical row-stores in which every row are stored one after another in the disk. These databases are more suitable for data warehousing system to get analysis done faster as data is stored in columnar form. Indexes are much faster in column oriented databases which results in faster data retrieval and hence data analysis. This is an alternate database technology over row oriented database systems.

There are two obvious ways to map database tables onto a one dimensional interface: store the table row-by-row or store the table column-by-column. The row-by-row approach keeps all information about an entity together. In the customer example above, it will store all information about the first customer, and then all information about the second customer, etc. The column-by-column approach keeps all attribute information together: the entire customer names will be stored consecutively, then all of the customer addresses, etc. Both approaches are reasonable designs and typically a choice is made based on performance expectations. If the expected workload tends to access data on the granularity of an entity (e.g., find a customer, add a customer, delete a customer), then the row-by-row storage is preferable since all of the needed information will be stored together.

On the other hand, if the expected workload tends to read per query only a few attributes from many records (e.g., a query that finds the most common e-mail address domain), then column-by-column storage is preferable since irrelevant attributes for a particular query do not have to be accessed (current storage devices cannot be read with fine enough granularity to read only one attribute from a row. The vast majority of commercial database systems, including the three most popular database software systems (Oracle, IBM DB2, and Microsoft SQL Server); choose the row-by-row storage layout. The design implemented by these products descended from research developed in the 1970s. The design was optimized for the most common database application at the time: business transactional data processing. The goal of these applications was to automate mission-critical business tasks. For example, a bank might want to use a database to store information about its branches and its customers and its accounts. Typical uses of this database might be to find the balance of a particular customer's account or to transfer $100 from customer A to customer B in one single atomic transaction. These queries commonly access data on the granularity an entity (find a customer, or an account, or branch information; add a new customer, account, or branch). Given this workload, the row-by-row storage layout was chosen for these systems.

Starting in around the 1990s, however, businesses started to use their databases to ask more detailed analytical queries. For example, the bank might want to analyze all of the data to find associations between customer attributes and heightened loan

ISSN 2278-7763

risks. Or they might want to search through the data to find customers who should receive VIP treatment. Thus, on top of using databases to automate their business processes, businesses started to want to use databases to help with some of the decision making and planning. However, these new uses for databases posed two problems. First, these analytical queries tended to be longer running queries, and the shorter transactional write queries would have to block until the analytical queries finished (to avoid different queries reading an inconsistent database state). Second, these analytical queries did not generally process the same data as the transactional queries, since both operational and historical data (from perhaps multiple applications within the enterprise) are relevant for decision making. Thus, businesses tended to create two databases (rather than a single one); the transactional queries would go to the transactional database and the analytical queries would go to what are now called data warehouses. This business practice of creating a separate data warehouse for analytical queries is becoming increasingly common;
In fact today data warehouses comprise $3.98 billion [65] of the $14.6 billion database market [53] (27%) and are growing at a rate of 10.3% annually [65].

Keywords: Column, database, performance, analytics, data warehouse, properties, attribute, data management, write oriented, implementation, implications, entity focused.

1. PROPERTIES OF ANALYTIC APPLICATIONS

The natures of the queries to data warehouses are different from the queries to transactional databases. Queries tend to be:

• **LESS PREDICTABLE.** In the transactional world, since databases are used to automate business tasks, queries tend to be initiated by a specific set of predefined actions. As a result, the basic structure of the queries used to implement these predefined actions is coded in advance, with variables filled in at run-time. In contrast, queries in the data warehouse tend to be more exploratory in nature. They can be initiated by analysts who create queries in an ad-hoc, iterative fashion.

• **LONGER LASTING.** Transactional queries tend to be short, simple queries ("add a customer", "find a balance", "transfer $50 from account A to account B"). In contrast, data warehouse queries, since they are more analytical in nature, tend to have to read more data to yield information about data in aggregate rather than individual records. For example, a query that tries to find correlations between customer attributes and loan risks needs to search though many records of customer and loan history in order to produce meaningful correlations.

• **MORE READ-ORIENTED THAN WRITE-ORIENTED.** Analysis is naturally a read-oriented endeavor. Typically data is written to the data warehouse in batches (for example, data collected during the day can be sent to the data warehouse from the enterprise transactional databases and batch-written over-night), followed by many read only queries. Occasionally data will be temporarily written for "what-if" analyses, but on the whole, most queries will be read-only.

• **ATTRIBUTE-FOCUSED RATHER THAN ENTITY-FOCUSED.** Data warehouse queries typically do not query individual entities; rather they tend to read multiple entities and summarize or aggregate them (for example, queries like "what is the average customer balance" are more common than "what is the balance of customer A's account"). Further, they tend to focus on only a few attributes at a time (in the previous example, the balance attribute) rather than all attributes.

2. IMPLICATIONS ON DATA MANAGEMENT

As a consequence of these query characteristics, storing data row-by-row is no longer the obvious choice; in fact, especially as a result of the latter two characteristics, the column-by-column storage layout can be better. The third query characteristic favors a column-oriented layout since it alleviates the oft-cited disadvantage of storing data in columns: poor write performance. In particular, individual write queries can perform poorly if data is 18 laid out column-by-column, since, for example, if a new record is inserted into the database, the new record must be partitioned into its component attributes and each attribute written independently. However, batch-writes do not perform as poorly since attributes from multiple

ISSN 2278-7763

records can be written together in a single action. On the other hand, read queries (especially attribute-focused queries from the fourth characteristic above) tend to favor the column-oriented layout since only those attributes accessed by a query need to be read, and thus this layout tends to be more I/O efficient. Thus, since data warehouses tend to have more read queries than write queries, the read queries are attribute focused, and the write queries can be done in batch, the column-oriented layout is favored.

Surprisingly, the major players in the data warehouse commercial arena (Oracle, DB2, SQL Server, and Teradata) store data row-by-row (in this dissertation, they will be referred to as row-stores"). Although speculation as to why this is the case is beyond the scope of this dissertation, this is likely due to the fact that these databases have historically focused on the larger transactional database market and wish to maintain a single line of code for all of their database software [64]. Similarly, database research has tended to focus on the row-by-row data layout, again due to the field being historically transitionally focused. Consequently, relatively little research has been performed on the column-by-column storage layout ("column-stores").

3. EVOLUTION

In the evolution of computing science, three generations of database technology are identified since the 60's till nowadays. The first generation started in the 60's and its main purpose was to enable disparate but related application to share data otherwise than passing files between them. The publishing of "A Relational Model of Data for Large Shared Data Banks" by E. F. Codd marked the beginning of the second generation of DBMS (database management systems) technology. Codd's premise was that data had to be managed in structures developed according to the mathematical set theory. He stated that data had to be organized into tuples, as attributes and relations.

A third generation began to emerge in the late 90's and now is going to replace second-generation products. Multi-core processors became common, 64-bit technology is used largely for database servers, memory is cheaper and disks are cheaper and faster than ever before. A recent IDC study examines emerging trends in DBMS technology as elements of the third generation of such technology. It considers that, at the current rate of development and adoption, the following innovations will be achieved in the next five years:

- Most data warehouses will be stored in a columnar fashion;
- Most OLTP (On-Line Transaction Processing) databases will either be augmented by an in-memory database or reside entirely in memory;
- Most large-scale database servers will achieve horizontal scalability through clustering;
- Many data collection and reporting problems will be solved with databases that will have no formal schema at all.

This study examines how some innovations in database technology field are implemented more and more. Most of these technologies have been developed for at least ten years, but they are only now becoming widely adopted. As Carl Olofson, research vice president for database management and data integration software research at IDC, said, "many of these new systems encourage you to forget disk-based partitioning schemes, indexing strategies and buffer management, and embrace a world of large-memory models, many processors with many cores,

Clustered servers, and highly compressed column wise storage". From the innovations that the study considers that will be achieved in the next years, this paper presents the columnar data storage.

4. COLUMN-ORIENTED DBMS

A column-oriented DBMS is a database management system (DBMS) that stores data tables as sections of columns of data rather than as rows of data, like most relational DBMSs. This has advantages for data warehouses, customer relationship management (CRM) systems, and library card catalogs, and other ad-hoc inquiry systems where aggregates are computed over large numbers of similar data items.

It is possible to achieve some of the benefits of column-oriented and row-oriented organization

ISSN 2278-7763

with any DBMSs. By denoting one as column-oriented, we are referring to both the ease of expression of a column-oriented structure and the focus on optimizations for column-oriented workloads. This approach is in contrast to row-oriented or row store databases and with correlation databases, which use a value-based storage structure.

A relational database management system must show its data as two-dimensional tables, of columns and rows, but store it as one-dimensional strings

A row-oriented database serializes all of the values in a row together, then the values in the next row, and so on.

A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on.

5. BASIC PROPERTIES OF A COLUMN-ORIENTED DATABASE

In highly replicated distributed systems, there is often a great emphasis on the durability of data and not just persistence. Durability requires that writes from a particular operation are stored in such a way that if an exception were to occur, that data can be recovered. The copies might be stored in memory or on disk, but they are typically written to a commit log. Durability can be achieved through replication as well, while persistence is focused on storing the data on disk for long term storage. Some column-oriented data stores, such as Dynamo, support pluggable storage engine architecture for persistence .Other data stores like Big Table and Cassandra utilize their own storage engine; both use SSTables.

Querying within a column-oriented data store is often limited to key only lookups, which are provided by each data store's own API. There is no query language, so data access is totally programmatic. Dynamo provides primary key only access, where every key in the Dynamo instance is unique since Dynamo does not provide a. Other data stores provide namespaces, such as column families and key spaces. These must be specified when querying data. To filter the data, some data stores such as Big Table allow for regular expressions to be passed via the query API call to reduce the number of rows that will be returned. Versioning techniques are critical to the concurrency model of column-oriented data stores. Updates within a row are commonly implemented as atomic operations with a timestamp used to denote the version. In some cases, the latest timestamp is the true version. Since there is no notion of isolation within column-oriented data stores, it is entirely possible that the latest timestamp is not the true version. In these situations, it is up to the client to resolve 13 version conflicts. In Dynamo terminology, this problem is referred to as semantic reconciliation. These reconciliation techniques are needed because there is generally no notion of a transaction in column-oriented data stores.

Security and access control is not a strong focus with column-oriented databases. This is an area where relational databases are much more robust. Dynamo, for example, expects to operate in a trusted and provides no security. Big Table, on the other hand, does support access control lists for column-families, which can be used to limit user capabilities. These access controls pale in comparison to row-level security, label based access control, and role based access control mechanisms supported by many mainstream relational databases.

In general, most popular column-oriented data stores place a lesser value on consistency and integrity compared to fault tolerance and low latency response. The alternative consistency model that these data stores focus on is called eventual consistency. To achieve high availability, replication amongst nodes is utilized extensively. Another wrinkle with respect to the integrity of column-oriented data stores is the fact that there is little or no support for types. Values are stored as uninterrupted byte strings, so it is up client applications in order to maintain consistent typing of values.

Support for recovery is effectively handled by using a commit log. Write operations are written to the commit log after finishing successfully. Without transactions, there is little support for rolling back operations and it is possible that a failure can occur during units of work. The primary goal of the commit log is to aid in providing durable storage for operations. Cassandra and Big Table log all writes to the commit log prior to updating their in-memory data structures or physical files.

ISSN 2278-7763

6. COLUMN-ORIENTED DATA MODEL IMPLEMENTATION

The column-oriented data model lends itself quite well to handling this sort of semi-structured data. Utilizing Cassandra's data model will provide a nested column-family structure for us to store our data. One choice for the logical data model would be to create a column family for each host, keyed by the host's IP address for their primary network interface. Each host would need a column key for hostname, a description of that host, and the MAC address for the primary physical network interface. This column family yield a list of the distinct hosts for which log messages will be stored.

Another column family could be used to store each log message for each host. Our key for this column family will be the concatenation of host IP address and the timestamp on the log 30 message. In this case, there is really no use for versioning each of the log messages, so a new instance of this column family will be created for each log message. For each message, we'll have column keys for every part that that will be parsed out of the original rsyslog formatted message. One key for each rsyslog property will be used along with a key for each part of the IPtables message.

Each protocol will have a slightly different list of keys. Cassandra supports this problem nicely by allowing us to add any number of column keys to a column family. This is in contrast with a relational data model where we would have to declare all possible message fields before hand, but only a subset of fields would be needed for each message. Given all the above requirements, the physical data model would be implemented as follows:

```
{
        "host_ip": {
                HOSTNAME:
                DESCRIPTION:
                MAC:
        }
}

{

        "host_ip"+"timestamp" {
                MSG:
                FROMHOST-IP:
                IN:
                OUT:
                MAC:
                SRC:
                DST:
                LEN:
                TOS:
                PREC:
                TTL:
                ID:
                PROTO:
                SPT:
                DPT:
                WINDOW:
                RES:
                FLAG:
                }
}
```

7. WHY COLUMN-ORIENTED DATABASES

The volume of data in an organization is growing rapidly. So does the number of users who need to access and analyse this data. IT systems are used more and more intensive, in order to answer more numerous and complex demands needed to make critical business decisions. Data analysis and business reporting need more and more resources. Therefore, better, faster and more effective alternatives have to be found. Business Intelligence (BI) systems are proper solutions for solving the problems above. Decision-makers need a better access to information, in order to make accurate and fast decisions in a permanent changing environment. As part of a BI system, reporting has become critical for a company's business.

Years ago, reports prepared by analysts were addressed only to the company's executive management. Nowadays, reporting has become an instrument addressed to decision-makers on all organizational levels, aiming to improve the company's activity, to ensure decision quality, control costs and prevent losses.

As already mentioned, the volume of data acquired into a company is growing permanently, because business operations expand and, on the other hand, the company has to interact with more sources of data and keep more data online. More than ever before, users need a faster and more

ISSN 2278-7763

convenient access to historical data for analysing purposes. Enterprise data warehouses are a necessity for the companies that want to stay competitive and successful. More and more reports and adhoc queries are requested to support the decision making process. At the same time,
Companies have to run audit reports on their operational and historical data in order to ensure compliance.

These new demands add more pressures upon IT departments. More and more hardware resources are needed in order to store and manage an increasing volume of data. The increasing number of queries needs larger amounts of CPU cycles, so more processors, having a higher performance, must be added to the system.

The size of the data warehouses storing this data is increasing permanently, becoming larger and larger. While five years ago the largest data warehouses were around 100 terabytes in size, now a data warehouse size at the petabyte level is no longer unusual. The challenge is to maintain the performance of these repositories, which are built, mostly, as relational structures, storing data in a row-oriented manner. The relational model is a flexible one and it has proven its capacity to support both transactional and analytical processing. But, as the size and complexity of data warehouses have increased, a new approach was proposed as an alternative on the row oriented approach, namely storing data in a column-oriented manner. Unlike the row oriented approach, where the data storage layer contains records (rows), in a column oriented system it contains columns. This is a simple model, more adequate for data repositories used by analytical applications, with a wide range of users and query types.

Researches indicate that the size of the largest data warehouse doubles every three years. Growth rates of system hardware performance are being overrun by the need for analytical performance. The volume of data needed to be stored is growing due to more and various requirements for reporting and analytics, from more and more business areas, increased time periods for data retention, a greater number of observations loaded in data warehouses and a greater number of attributes for each observation. This is true if taking into consideration only structured data. But nowadays, organizations collect a larger and larger volume of unstructured data, as images, audio and video files, which need a much greater storing space than structured data.

Row-oriented databases have been designed for transactional processing. For example, in the account management system of a bank, all attributes of an account are stored in a single row. Such an approach is not optimal in an analytical system, where a lot of read operations are executed in order to access a small number of attributes from a vast volume of data. In a row-oriented architecture, system performance, users' access and data storage become major issues very quickly. As they are designed to retrieve all elements from several rows, row oriented databases are not well suited for large scale processing, as needed in an analytical environment. As opposed to transactional queries, analytical queries typically scan all the database's records, but process only a few elements of them. In a column-oriented database all instances of a single data element, such as account number, are stored together so they can be accessed as a unit. Therefore, column oriented databases are more efficient in an analytical environment, where queries need to read all instances of a small number of data elements.

System performance enhances spectacularly in a column-oriented solution, because queries search only few attributes, and they will not scan the attributes that are irrelevant for those queries. Requested data is found faster, because less sort operations have to be performed.

A typical feature of evolved BI systems is their capability to make strategic business analyses, to process complex events and to drill deeply into data. As the volume of data becomes impressive and performance demands required by users are likely to outpace, it is obviously that row-oriented relational database management systems stopped to be the solution for implementing a BI system having powerful analytical and predictive capabilities. A new model tends to come into prominence as an alternative on developing analytical databases, namely one that manages data by columns.

A column-oriented DBMS stores data in a columnar manner and not by rows, as classic DBMS do. In the columnar approach, each attribute is stored in a separate table, so successive

ISSN 2278-7763

values of that attribute are stored consecutively. This is an important advantage for data warehouses where, generally, information is obtained by aggregating a vast volume of data. Therefore, operations as MIN, MAX, SUM, COUNT, AVG and so forth are performed very quickly.

When the tables of a database are designed, their columns are established. The number of rows will be determined when the tables will be populated with data. In a row oriented database, data is stored in a tabular manner. The data items of a row are stored one after another; rows are also stored one after another, so the last item of a row is followed by the first item of the next row.

In a column-oriented database, the data items of a column are stored one after another, and also are the columns; so the last item of a column is followed by the first item of the next column.

8. BENEFITS

Comparisons between row-oriented and column-oriented data layouts are typically concerned with the efficiency of hard-disk access for a given workload, as seek time is incredibly long compared to the other delays in computers. Sometimes, reading a megabyte of sequentially stored data takes no more time than one random access.[3] Further, because seek time is improving much more slowly than CPU power (see Moore's Law), this focus will likely continue on systems that rely on hard disks for storage. Following is a set of oversimplified observations which attempt to paint a picture of the trade-offs between column- and row-oriented organizations Unless, of course, the application can be reasonably assured to fit most/all data into memory, in which case huge optimizations are available from in-memory database systems.

1. Column-oriented organizations are more efficient when an aggregate needs to be computed over many rows but only for a notably smaller subset of all columns of data, because reading that smaller subset of data can be faster than reading all data.
2. Column-oriented organizations are more efficient when new values of a column are supplied for all rows at once, because that column data can be written efficiently and replace old column data without touching any other columns for the rows.
3. Row-oriented organizations are more efficient when many columns of a single row are required at the same time, and when row-size is relatively small, as the entire row can be retrieved with a single disk seek.
4. Row-oriented organizations are more efficient when writing a new row if all of the column data is supplied at the same time; as the entire row can be written with a single disk seek.
5. Advantage of column oriented databases over row oriented databases is in the efficiency of hard- disk access

9. EXAMPLES OF COLUMN-ORIENTED DATABASE SYSTEMS

SYBASE IQ: Sybase IQ is a high-performance decision support server designed specifically for data warehousing. It is a column oriented relational database that was built, from the very beginning, for analytics and BI applications, in order to assist reporting and decision support systems. This fact offers it several advantages within a data warehousing environment, including performance, scalability and cost of ownership benefits.

VERTICA: Vertica Analytic Database is a DBMS that can help in meeting these needs. It is a column-oriented database that was built in order to combine both column store and execution, as opposed to other solutions that are column-oriented only from storage point of view. Designed by Michael Stonebraker, it incorporates a combination of architectural elements – many of them which have been used before in other contexts – to deliver a high-performance and low-cost data warehouse solution that is more than the sum of its elements.

10. CONCLUSIONS:

For applications that write and update many data (OLTP systems), a row-oriented approach is a proper solution. In such architecture, all the attributes of a record are placed contiguously in storage and are pushed out to disk through a

ISSN 2278-7763

single write operation. An OLTP system is a write optimized one, having a high writing performance. In contrast, an OLAP system, mainly based on ad-hoc queries performed against large volumes of data, has to be read optimized. The repository of such a system is a data warehouse. Periodically (daily, weekly, or monthly, depending upon how current data must be), the data warehouse is load massively. Ad-hoc queries are then performed in order to analyse data and discover the right information for the decision making process. And for analytical applications, that read much more than they write a column-oriented approach is a better solution.

Nowadays, data warehouses have to answer more and more ad-hoc queries, from a greater number of users which need to analyse quickly larger volumes of data. Columnar database technology inverts the database's structure and stores each attribute separately, fact that eliminates the wasteful retrieval as queries are performed.

On the other hand, much more data can be loaded in memory, and processing data into memory is much faster. Column-oriented databases provide faster answers, because they read only the columns requested by users' queries, since row-oriented databases must read all rows and columns in a table. Data in a column oriented database can be better compressed than those in a row-oriented database, because values in a column are much more homogenous than in a row. The compression of a column-oriented database may reduce its size up to 20 times, this thing providing a higher performance and reduced storage costs. Because of a greater compression rate, a column-oriented implementation stores more data into a block and therefore more data into a read operation. Since locating the right block to read and reading it are two of the most expensive computer operations, it's obviously that a column-oriented approach is the best solution for a data warehouse used by a Business Intelligence system developed for analytical purposes.

data warehouse system used for data analysis and reporting.

References:-

1. S.G.Yaman on Introduction to Column-Oriented Database Systems at cs.helsinki.fi
2. A Review of Column-oriented Data stores by Zach Pratt at attackofzach.com
3. Column-Oriented Databases, an Alternative for Analytical Environment by Gheorghe MATEI at dbjournal.ro
4. wikipedia.org and wikimedia.org

11. ACKNOWLEDGMENT

I thanks Dr. A.K. Yadav to encourage to research and write on Column based database and its importance when we deal in large database like