# A COMPREHENSIVE STUDY OF EXECUTION TIME FOR ROW AND COLUMN ORIENTED DATABASE

Mr. H.K. Mishra[1], Mr.Yashwant Soni[2], Ms. Sakshi Gautam[3]
[1]HOD in EE Deptt. Sobhasaria group of institutions, Sikar,hare.dbit@gmail.com
[2]Asst. Prof. in CSE Deptt. Sobhasaria group of institutions, Sikar,yashalw80@gmail.com
[3]Asst. Prof. in CSE Deptt. JBIT, Dehradun , 1sakshigautam@gmail.com.

**Abstract--**The size of data in various types of databases are increasing rapidly. At the same time, the performances of query against these same databases are degrading. There are two methods to implement a two-dimension relational database table onto a one-dimensional storage interface: store the table row-by-row, or store the table column-by-column. Traditionally, database system implementations and research have focused on the row-by row data layout, since it performs best on the most common application for database systems: business transactional data processing. However, there are a set of emerging applications for database systems for which the row-by-row layout performs poorly. The need for Column-oriented database arose from the need of business intelligence needed for efficient decision making where traditional Row-oriented database gives poor performance. We know that Business organizations have to handle large amount of data in database and extract meaningful information from that database for efficient decision making which is commonly termed as Business Intelligence. Extracting meaningful information from raw data is term as data mining. In this paper, we study the poor performance of row-by-row data layout for these emerging applications, and evaluate the column-by-column data layout opportunity as a solution to this problem. The solution will be analyzed and represented by graph. At the end of the paper we will see the performance of Oracle 10g.

**Keywords--**Databases, Database Systems, Row Store, Column Store, Performance Tuning

## 1. INTRODUCTION

Whenever we say relational data, most obvious interpretation is a table which has attribute as one dimension and entity as another. We imagine a table stored on some storage media in such a 2-dimensional form. But this is just a concept for better understanding of any relation stored some storage media. At physical level, it is not possible to store data like the way we imagine. Therefore, Data are physically stored consecutively one after another in 1-dimensional way. While storing in 1-dimensional manner we have 2 choices. We can either store the data entity-by-entity or attribute-by-attribute. This leads to two kinds of databases Row-Store and Column-Store respectively.

### 1.1 Rows v/s Columns

The question of which type of database system is better depends on the kind of query workloads . If after data insertion, updation, deletions are going to be more and if accessing entire tuples is a need then Row-Stores are the best. They are the most common ones for business transactional data processing. For example, a bank uses databases to store information of its customers. Some customer A might want to transfer money to the account of customer B. Here, Customer A and B are entities. Here a simple updation has to be done in accounts of A and B both which is deduct amount x from account of A and credit amount x to account of B. As it can be seen information will be required by the bank from DBMS on the granularity of an entity here, Row-Store which stores data entity-by-entity will be most obvious choice out of the two database systems we studied. If we consider another query, customers shopping for more than Rs. 5000 every month,(Owner thinks if additional benefits are given to these customers then they might visit the shop more often). This query needs only customer name, amount spent and date attributes from the entire relation. Clearly, rest 10-15 attributes will be irrelevant (assuming dataset is very large). This query will help to gain insight into the data and it is not business critical situation like in transactional processing. For such kind of queries Column-Stores perform better since attributes are stored separately so irrelevant attributes need not be accessed saving a lot of processing time. Suppose if queries are going to be Read queries which will help to gain insight into the data, Column-Stores will certainly perform better. Therefore, when it comes to analytical applications or decision making applications, column-stores prove to be the best [3]. Business organizations have to handle large amount of data and extract meaningful information from that data for efficient decision making which is commonly termed as Business Intelligence.

Again there are some optimizations possible with Column-Stores and are not possible with Row-Stores which can improve performance of Column-Stores compared Row-Stores significantly [2, 3]. The rest and the most important is Compression [8]. As data are stored column-by-column, compression can be easily applied on a column. This is possible because a column has a data type in which similar data is stored. Like mobile number in India will always contain 10 digits. If one could store data is compressed format, performing column extraction will become very easy. Next is block processing, where multiple tuples from a column are extracted and are passed as a block from one operator to another. There is one more optimization called as Late Materialization where tuples construction i.e. joining of columns is performed as late as possible. These optimizations are specific to Column-Stores because Row-Stores do not have required properties to apply these optimizations.

## 2. BACKGROUND AND RELATED WORK

A relational database management system provides represents data into a two-dimensional table, which consist of columns and rows. Row-based systems are not efficient at performing operations that apply to the entire data set, as opposed to a specific record.[2,3,4]

In our work, we see that previously there are various approaches are implemented for Column-Store database and I found that Vertical Partitioning is most preferred of all due to less

complexity and no limitations on the kind of possible read queries.
In this section we are showing that what are the disadvantages of row oriented database and how we can improve the performance of sql query with column oriented database techniques.

### 2.1Merit of column store

- Improved bandwidth utilization: In a column-store only those attribute that is accessed by a query needs to be read-off disk (or from memory into cache). In a row store surrounding attributes also need to read since the attribute is generally smaller than the smallest granularity in which data can be accessed.
- Improved data compression: Storing data from the same attribute domain increases locality and thus data compression ratio (especially if the attribute is sorted). Bandwidth requirements are further reduced when transferring compressed data [1,8]
- Improved code pipelining: Attribute data can be iterated through directly without indirection through a tuple interface. This results in high IPC (instructions per cycle) efficiency, and code that can take advantage of the super-scalar properties of modern CPUs [4, 5].

- Improved cache locality: A cache line also tends to be larger than a tuple attribute, so cache lines may contain irrelevant surrounding attributes in a row-store. This wastes space in the cache and reduces hit rates [6].

## 2.2 Demerit of column-stores:

- Increased disk seek time: Disk seeks between each block read might be needed as multiple columns are read in parallel. However, if large disk pre-fetches are used, this cost can be kept small[8]
- Increased cost of inserts: Column-stores perform poorly for insert queries since multiple distinct locations on disk have to be updated for each inserted tuple (one for each attribute). This cost can be alleviated if inserts are done in bulk.
- Increased tuple reconstruction costs: In order for column-stores to offer a standards-compliant relational database interface (e.g., ODBC, JDBC, etc.), they must at some point in a query plan stitch values from multiple columns together into a row-store style tuple to be output from the database. Although this can be done in memory, the CPU cost of this operation can be significant. In many cases, reconstruction costs can be kept to a minimum by delaying construction to the end of the query plan [9].

## 3. IMPLEMENTATION

Our goal is to design column oriented databases and to propose new ideas for performance optimization. One approach of implementing column oriented database is to vertically partition a traditional row oriented database. Tables in the row store are broken up into multiple two column tables consisting of (table key, attribute) pairs. There is one two column tables for each attribute in the original table. When a query is issued, only those thin attribute-tables relevant for a particular query need to be accessed-the other tables can be ignored. These tables are joined on table key to create projection of original table containing only those columns necessary to answer a query, and then execution proceeds as normal. The smaller the percentage the columns from table that need to be accessed to answer a query the better the relative performance with a row store will be.
In a fully vertically partitioned approach, some mechanism is needed to connect fields from the same row to together (column stores typically matchup records implicitly by storing columns in the same order, but such optimization are not available in a row store). To accomplish this, the simplest approach is to add an integer "position" column to every table- this is often preferable to use the primary key because primary keys can be large and are sometimes composite. This approach creates one physical table for each column in the logical schema. By the example given below the conversion of a row by row database to column oriented database can be shown.
For performance analysis of row oriented database vs column oriented database there is a need of large row-oriented database. Using this large row-oriented database column-oriented database can be derived by vertical partitioning. Analysis of performance will be based on execution time of sql queries on the row oriented database and column oriented respectively. In this paper Oracle 10g is taken as database software.
There are two table in the database name Account_Table(Branch_Name,Account_Number, Balance)) and Depositor_Table (Cust_Name,Account_Number). Initially both tables contains million records each. By Vertical partitioning on the given tables we derived new tables and a separate database has been made. The tables are
**Account**_X (SNO,Branch_Name), **Account**_Y(SNO,Account_Number), Account_Z(SNO,Balance), Depo_1(SNO Cust_Name) and Depo_2 (SNO,Account_Number), respectively.
Now we have take the internal join of any two or three table according to requirement queries will execute on this database and the performance will analyzed on the basis of query execution time(in sec).

## 3.1 Analysis of Performance

Performance will be analyzed on software Oracle 10g .
Let us see what difference does this approach make in the query plan of a
SELECT query which is as follows:

**For row store**

- select count (distinct cust_ name) from Depositor, Account where Depositor . account _number = Account . account _number and Branch_ name = 'brighton' group by branch_name;

**For column store**

- select count ( distinct cust_name) from depo_1, Account_X where depo_1.srno = Account_X.srno and Branch_name='brighton'
group by Branch_name;
- In this SELECT query, Depo_1.srno , Account _X.srno and Branch_Name are predicates. Predicate is a attribute present in a query on which some condition is applied. Also, Cust_name is non-predicates. Non-

predicate is an attribute present in the query which is to be projected. Hence, these attributes are considered while taking natural join for corresponding tables. Here, two natural joins of internal tables will be taken. One will be for Depositor table (Cust_Name and Account_Number ) and another for Account table 1 (Account_Number and Branch_Name).

- Now let us see the performance comparison of Row-Store against Column- Store with the help of some sql queries with following result.

Table1: Experimental results for simple select query

| **Sequence of query execution** | **Execution Time in seconds For Row-Store** | **Execution Time in seconds For Column-Store** |
|---|---|---|
| 1 | 0.45 | 0.16 |
| 2 | 0.03 | 0.02 |
| 3 | 0.11 | 0.02 |
| 4 | 0.02 | 0.01 |
| 5 | 0.08 | 0.06 |
| 6 | 0.08 | 0.02 |
| 7 | 0.02 | 0.01 |
| 8 | 0.03 | 0.02 |
| 9 | 0.02 | 0.01 |
| 10 | 0.05 | 0.03 |

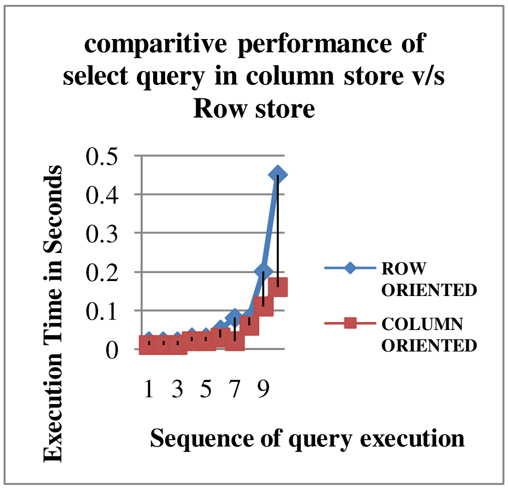

Figure1: Query Performance on Oracle10g database

By Fig it is clear that Column Oriented database performs better than Row-Oriented database at certain conditions on Oracle 10g.

## 5. CONCLUSION

In our work, we investigated various approaches of implementation of Column-Store .The results show that performance of our Column-Store implementation is very high as compared to Row-Store in queries. Using Column-Stores only attributes which are present the select query as a predicate or non-predicate, are accessed which reduces execution time as compared to that in Row-Stores [8]. This concept is implemented for Column-Store implementation in oracle10g. We see that as number of columns accessed increases, the performance of Column-Store degrades which is as expected. This is because number of joins of internal tables increases in such a case which leads to increase in execution time. The same case would be very efficient in Row-Store. But, the idea behind Column-Stores is to use them for specific applications such as data mining, data are housing and scientific datasets. Vertical partitioning approach to build a column-store requires slight modifications in the DBMS. This modification in the DBMS will certainly result is significant performance gains for large databases. It will certainly be useful for data warehouses where the analysis is naturally a read oriented endeavor. Unlike row oriented databases write optimized nature column oriented databases will be read optimized. Vertical partitioning is a good approach for column oriented database design but this approach also introduces extra redundancy in the database. So instead of using primary key or serial no indexing can be used. In future I want to implement data directly into column manner in which write query can give better performance.

## 6. REFERENCES

[1] Raichand priyanka and aggarwal rinkle rani department of cse, thapar university, patiala short survey of data compression techniques for column oriented databases volume 4, no. 7, july 2013 journal of global research in computer science.

[2]Dwivedi Amit Kumar ,Lamba C. S. ,Shukla Shweta , Performance Analysis of Column Oriented Database Vs Row Oriented Database, IJCA Journal Volume 50 Number 14 ,2012,

[3] Daniel J. Abadi, Samuel R. Madden, Nabil Hachem, “Column-Stores vs Row-Stores : How Different Are They Really?”, Vancouver, BC, Canada, SIGMOD’08, (2009-12)

[4] Stavros Harizopoulos (HP Labs), Daniel Abadi (Yale), Peter Boncz (CWI), “Column-Oriented Database Systems”, VLDB Tutorial, (2009).

[5]D. J. Abadi, “Query execution in column-oriented database systems”, MIT PHD Dissertation(2008).

[6] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, "Materialization strategies in a column-oriented DBMS", In ICDE, pp 466-475, (2007).

[7] G. Graefe, "Efficient columnar storage in b-trees". SIGMOD Rec., 36(1):pp 3–6 ,2007

[8]D.J.Abadi,S. R. Madden, and M. Ferreira,"Integrating compression and execution in column oriented database systems", In SIGMOD, pp 671-682, 2006

[9] S.Harizopoulos, V. Liang, D. J. Abadi, and S.R. Madden. "Performance tradeoffs in readoptimized databases", VLDB, pp 487–498, 2006

[10] Halverson, J. L. Beckmann, J. F. Naughton, and D. J. Dewitt, "A Comparison of C-Store and Row-Store in a Common Framework." Technical Report TR1570, University of WisconsinMadison,2006