Scientific Journal of Impact Factor (SJIF): 4.72

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 4, Issue 7, July -2017

SURVEY AND COMPARISON OF ROW /COLUMN ORIENTED DATABASE AND IMPROVED THE PERFORMANCE OF EXECUTION TIME FOR A SELECT QUERY

[1]Mr.Mohit Sani, [2]Mr. Yashwant Soni, [3]Mr. Sandeep Srivastava

[1]*Sobhasaria group of institutions, Sikar*
[2]*Asst. Prof. in CSE Deptt.Sobhasaria group of institutions, Sikar*
[3]*Asst. Prof. in CSE Deptt.Sobhasaria group of institutions, Sikar*

ABSTRACT:- *The size of data in varied sorts of databases ar increasing speedily. At an equivalent time, the performances of question against these same databases ar degrading. There ar 2 strategies to implement two-dimension electronic information service table onto a one-dimensional storage interface: store the table row-by-row, or store the table column-by-column. Historically, information system implementations and analysis have centered on the row-by row data layout, since it performs best on the foremost common application for information systems: business transactional processing. However, there ar a group of rising applications for information systems that the row-by-row layout performs poorly. the requirement for Column-oriented information arose from the requirement of business intelligence required for economical deciding wherever ancient Row-oriented information provides poor performance. we all know that Business organizations got to handle great deal of {information} in information and extract significant information from that information for economical deciding that is usually termed as Business Intelligence. Extracting significant data from data is term as data processing. during this paper, we have a tendency to study the poor performance of row-by-row data layout for these rising applications, and assess the column-by-column data layout chance as an answer to the present drawback. the answer are going to be analyzed and diagrammatical by graph. At the top of the paper we'll see the performance of Oracle 10g.*

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

1. INTRODUCTION

Whenever we are saying relative data, most evident interpretation could be a table that has attribute jointly dimension and entity as another. we tend to imagine a table keep on some storage media in such a 2-dimensional type. however this is often simply an idea for higher understanding of any relation keep some storage media. At physical level, it's inconceivable to store data just like the approach we tend to imagine. Therefore, data square measure physically keep consecutively one once another in 1-dimensional approach. whereas storing in 1-dimensional manner we've a pair of decisions. we will either store the info entity-by-entity or attribute-by-attribute. This ends up in 2 forms of databases Row-Store and Column-Store severally.

1.1 Rows v/s Columns

The question of which sort of info system is best depends on the type of question workloads . If once knowledge insertion, updation, deletions ar getting to be a lot of and if accessing entire tuples could be a want then Row-Stores ar the most effective. they're the foremost common ones for business transactional processing. as an example, a bank uses databases to store info of its customers. Some client A would possibly need to transfer cash to the account of client B. Here, client A and B ar entities. Here an easy updation must be worn out accounts of A and B each that is deduct quantity x from account of A and credit quantity x to account of B. because it may be seen info are needed by the bank from package on the roughness of Associate in Nursing entity here, Row-Store that stores knowledge entity-by-entity are most blatant alternative out of the 2 info systems we have a tendency to studied. If we have a tendency to contemplate another question, customers buying over Rs. 5000 monthly,(Owner thinks if further edges ar given to those customers then they may visit the search a lot of often). this question desires solely client name, quantity spent and date attributes from the complete relation. Clearly, rest 10-15 attributes are tangential (assuming dataset is incredibly large). this question can facilitate to realize insight into the information and it's not business important scenario like in transactional process. 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.

2.1 Merit of column store

a) Improved information measure utilization: during a column-store solely those attribute that's accessed by a question has to be read- off disk (or from memory into cache). during a row store close attributes conjointly got to scan since the attribute is usually smaller than the tiniest roughness during which knowledge is accessed.
b) Improved knowledge compression: Storing knowledge from constant attribute domain will increase neighborhood and therefore knowledge compression magnitude relation (especially if the attribute is sorted). information measure necessities area unit additional reduced once transferring compressed knowledge [1,8]
c) Improved code pipelining: Attribute knowledge is iterated through directly while not indirection through a tuple interface. This leads to high IPC (instructions per cycle) potency, and code that may cash in of the super-scalar properties of contemporary CPUs [4, 5].
d) Improved cache locality: A cache line conjointly tends to be larger than a tuple attribute, therefore cache lines could contain unsuitable close attributes during a row-store. This wastes area within the cache and reduces hit rates [6].

2.2 Demerit of column-stores:

a) enlarged disk look for time: Disk seeks between every block browse may be required as multiple columns area unit browse in parallel. However, if massive disk pre-fetches area unit used, this price will be unbroken small[8]
b) Increased price of inserts: Column-stores perform poorly for insert queries since multiple distinct locations on disk have to be compelled to be updated for every inserted tuple (one for every attribute). This price will be mitigated if inserts area unit tired bulk.
c) *enlarged tuple reconstruction costs: so as for column-stores to supply a standards-compliant computer database interface (e.g., ODBC, JDBC, etc.), they need to at some purpose in a very question set up sew values from multiple columns along into a row-store vogue tuple to be output from the info. though this could be tired memory, the electronic equipment price of this operation will be vital. In several cases, reconstruction prices will be unbroken to a minimum by delaying construction to the tip of the question set up [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 an 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

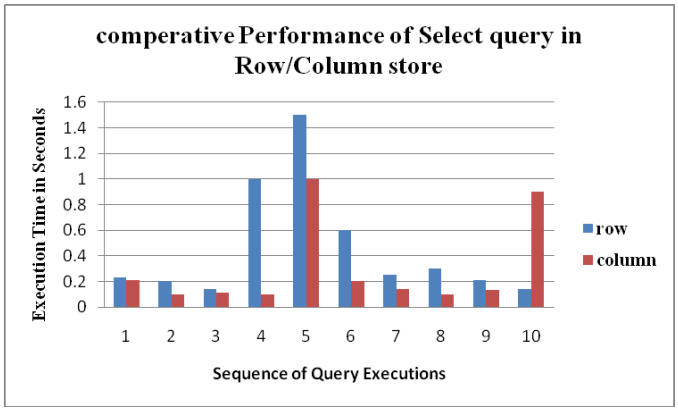

Figure1: Query Performance on Oracle10g database

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 have a tendency to investigated varied approaches of implementation of Column-Store .The results show that performance of our Column-Store implementation is extremely high as compared to Row-Store in queries. victimization Column-Stores solely attributes that square measure gift the choose question as a predicate or non-predicate, square measure accessed that reduces execution time as compared thereto in Row-Stores [8]. this idea is enforced for Column-Store implementation in oracle10g. we have a tendency to see that as range of columns accessed will increase, the performance of Column-Store degrades that is of course. this can be as a result of range of joins of internal tables will increase in such a case that results in increase in execution time. an equivalent case would be terribly economical in Row-Store. But, the thought behind Column-Stores is to use them for specific applications like data processing, information square measure housing and scientific datasets. Vertical partitioning approach to make a column-store needs slight modifications within the software system. This modification within the software system will definitely result's important performance gains for giant databases. it'll actually be helpful for information warehouses wherever the analysis is of course a browse adjusted endeavor. in contrast to row adjusted databases write optimized nature column adjusted databases are going to be browse optimized. Vertical partitioning may be a smart approach for column adjusted information style however this approach conjointly introduces further redundancy within the information. therefore rather than victimization primary key or serial no compartmentalization will be used. In future i would like to implement information directly into column manner within which write question will offer higher 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.**