A Comparitive Study of Indexing Techniques in Data Warehouse

Neha Sharma[1] and Arvind Panwar[2]
[1,2]Northern India Engineering College, Delhi, India

***Abstract*— The need for strategic information is escalating day by day and that can be fulfilled efficiently with the help of data warehouse indexing techniques. For this, one needs to maintain the data warehouse structure well. Data-warehouse has a broader scale and is completely equipped with the decision making process strategies. Since data in the data warehouse is enormous, so most of the queries in such a data warehouse will be complex and time taking too. Therefore, to speedup the processing time of these queries, indexing is used. At present, there are a number of indexing techniques available in the market. But according to the cardinality, size, processing speed, flexibility, and ease to make etc. different indexing techniques are visualized. This paper provides an assessment of different indexing techniques of data warehouse indexing depending upon above defined criteria's.**

***Keywords*—** Data Warehouse, Techniques and Indexing

I. INTRODUCTION

In today's scenario, each and every company needs strategical information for the purpose of business analysis to look at the prevailing information crisis. This information needs to be gathered as quickly as possible for the business analysis. Hence, to speedup the query processing in data warehousing environment, indexing is used and this paper focuses on explanation of different indexing techniques and their use. It also emphasis on use of different indexing techniques in different scenarios and it presents a comparison between all of them.

Data-warehouse is a subject oriented, integrated, non-volatile and time variant collection of data in support of management's decision [1], [2]. Data in the data warehouse is a compilation of information gathered from Online Transaction Processing (OLTP). Slight modifications are done on traditional OLTP systems and the new thing named Online Analytical Processing (OLAP) comes into the picture. Data in the data warehouse is presented in the form of multidimensional model. OLAP tools facilitate users to scrutinize multidimensional data from numerous perspectives by allowing query the system iteratively to take faster and better decisions. Generally, these queries take long time to execute because of number of join operations on number of records.

These complex queries will include complex functions such as "group by", "having", and "order by" and this will result in number of days to compute result of these queries. Queries defined on data warehouses are generally analytical and multifarious, which uses a number of join operations. Join operations are very costly, particularly when run on very large data volumes. A majority of requests for information from a data warehouse involve dynamic ad hoc queries users can ask any question at any time for any reason against the base table in a data warehouse. The ability to answer these queries quickly is a critical issue in the data warehouse environment [3], [4].

To overcome this problem of slow query processing and to improve response time, the concept of indexing came. Data warehouse reports needs quick response. Hence, many indexing techniques have been created to accomplish this target. So indexing enhances the ability to extract data to answer complex and ad hoc queries speedily. A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records [5].

Different types of indexing techniques explained in this paper are listed below:

- B-Tree index,
- Pure Bitmap index,
- Encoded bitmap index,
- Bitmap Join Index, and
- Projection index.

The above named indexing techniques will be explained in detail in the section named Indexing Structure and comparison of these techniques will be presented in the section Indexing Comparative study of indexing techniques.

II. LITERATURE REVIEW

Vanichayobon et al provides an evaluation of indexing techniques being studied/used in both academic research and industrial applications [6]. In addition, it also identifies the factors that need to be considered when one wants to build a proper index on base data. The objectives of this paper are to identify factors that need to be considered in order to select a proper indexing technique for data warehouse applications, and to evaluate these indexing techniques.

Maher et al., describes the Simple Bitmap Index technique and its related techniques such as Bit-Sliced Index, Range-Based Index and Encoded Bitmap Index [7] and another Indexing technique Projection Index. It also compares those

Indexes techniques concerning many parameters. Finally, it presents a conclusion and future enhancement of the Bitmap Indexes. The main focus of this paper is bitmap indexing. It emphasizes on bitmap indexing types, advantages, disadvantages, application and then at last it shows how bitmap indexing is different from other indexing techniques by a simple comparison between al of them.

Ghanjaoui et al., investigate the usability and performance of the UB-Tree (Universal B-Tree) as one of these multidimensional access methods [8]. The UB-Tree is a very promising multidimensional index, which has shown superiority over traditional access methods in different scenarios, especially in OLAP applications. It is being shown that the advantages of the UB-Tree in performance and cache requirements are Considerable. In addition, the UB-Tree can be integrated into existing database systems relatively easy, as most of its operations are based on the B-Tree. A real example is the integration of the UB-Treeinto the commercial Database Management System (DBMS) TransBase. This project was awarded the 2001 IT-Prize by EUROCASE and the European commission.

Weigel reviews fourteen approaches to indexing semi structured data, ranging from most simple look-up tables as used in traditional Information Retrieval to sophisticated graph-based index structures for combined structure and content queries [9]. Each approach is described in detail, including illustrated examples. For comparison, they emphasize differences in the employed data structures, the document types supported, the kind of information to be retrieved (specified in terms of look-up input and output), the storage and update behavior, and performance results. Moreover, specific features such as structural node identification, path templates, and internal navigation are examined. The major preliminaries from Information Retrieval, structured documents, and indexing in general are discussed in three introductory chapters. They include generic retrieval system architecture and further data structures such as schema graphs, signatures, filters, and various Tries.

Azefack et al present an automatic, dynamic index selection method for data warehouses that is based on incremental frequent item set mining from a given query workload [10]. The main advantage of this approach is that it helps update the set of selected indexes when workload evolves instead of recreating it from scratch. Preliminary experimental results illustrate the efficiency of this approach, both in terms of performance enhancement and overhead.

III. THE INDEXING STRUCTURE

Indexes are basically database objects, which are used to speed up the data retrieval from huge databases. Indexing is an old technique and it exists in relational database from past so many years but it's not efficient on large number of data records as present in data warehouse, which is used for strategic analysis.

A. Bitmap Indexing

Bitmap indexes are principally meant for data warehousing applications where there's a large amounts of data and ad hoc queries based on that huge data. In a bitmap index, a bitmap is used as a key value to replace list of row ids. Each bit of the bitmap represents a possible row id, and if the bit is set ('1'), it means the row with the corresponding row id contains the key value. A mapping function is used to convert the bit position to an actual row id. As a result, the bitmap index works similar to a regular index.

Following are major advantages of using Bitmap Index:

- Reduced response time for large classes of ad hoc queries
- Reduced storage requirements compared to other indexing techniques
- Dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory
- Efficient maintenance during parallel DML and loads

1) Pure Bitmap Index

Pure Bitmap Index was first introduced and implemented in the Model 204 DBMS. It consists of a collect of bitmap vectors each of which is created to represent each distinct value of the indexed column. A bit ***i*** in a bitmap vector, representing value ***x***, is set to 1 if the record ***i*** in the indexed table contains ***x***.

Following are merits of using pure Bitmap Index:

- It is well suited for low cardinality columns.
- It utilizes bitwise operations.
- The indexes can be combined before fetching raw data.
- It uses low space; it works well with parallel machine.
- It is easy to build.
- It performs efficiently with columns involving scalar functions (e.g., COUNT).
- It is easy to add new indexed value.
- It is suitable for OLAP.

Following are demerits of using pure Bitmap Index:

- It performs inefficiently with high cardinality data.
- It is very expensive when we update index column. The whole bitmap segment of the updated row is locked so the other row can not be updated until the lock is released.
- It does not handle spare data well.

2) Encoded Bitmap Index

An Encoded Bitmap Index on a column ***A*** of a table ***T*** consists of a set of bitmap vectors, a lookup table, and a set of retrieval Boolean functions. Each distinct value of a column A is encoded using a number of bits each of which is stored in a bitmap vector. The lookup table stores the mapping between A and its encoded representation. IBM implements this index in DB2.

Comparing with the Pure Bitmap Index, the Encoded Bitmap Index improves the space utilization, and solves sparsity problems. The size of the Encoded Bitmap Index built on the high cardinality column is less than the Pure Bitmap Index. Having a well defined encoding scheme, a Boolean operation can perform on the retrieval functions

before retrieving the data, and lead to a reduction of the number of bitmap vectors read. Its performance is degraded with equality queries since we have to search all the bitmap vectors. The index needs to be rebuilt if we run out of bits to represent new values.

Advantages of using encoded Bitmap Index are:

- It uses space efficiently.
- It performs efficiently with wide range query [8].

Disadvantages of using encoded Bitmap Index are:

- It performs inefficiently with equality queries.
- It is very difficult to find a good encoding scheme.
- It is rebuilt every time when a new indexed value for which we run out of bit to represent is added [8].

3) Bitmap Join Index

A Join Index is built by translating restrictions on the column value of a dimension table (i.e., the gender column) to restrictions on a large fact table. The index is implemented using one of the two representations: row id or bitmap [11], depending on the cardinality of the indexed column. A bitmap representation, which is called Bitmap Join Index, is used with the low cardinality data while a row id representation is used with a high cardinality. In DW, there are many join operations involved; so building Join Indexes on the joining columns improves query-processing time.

Advantages of using Bitmap join Index are:

- It is flexible.
- It performs efficiently.
- It supports star queries.

Disadvantages of using Bitmap join Index are:

- The order of indexed column is important [8].

B. B-tree index

B-tree indexing is an indexing technique that has a concerned awareness in multi dimensional database performance. A B-tree index is organized like an inverted tree. The leaves of the index hold the actual data values and pointers to the subsequent rows. In a data warehouse, B-tree indexes should be used only for unique columns or other columns with very high cardinalities that is, columns that are almost unique. B-tree indexes can be used when a query refers to the indexed column and retrieves a few rows. In these queries, it is faster to find the rows by looking at the index. Merits of using B-Tree indexing are:

- It speeds up known queries.
- It is well suited for high cardinality.
- The space requirement is independent of the cardinality of the indexed column.
- It is relatively inexpensive when we update the indexed column since individual rows are locked [8].

Demerits of using B-Tree indexing are:

- It performs inefficiently with low cardinality data.
- It does not support ad hoc queries. More I/O operations are needed for a wide range of queries.
- The indexes cannot be combined before fetching the data [8].

C. Projection Index

A Projection Index on an indexed column ***A*** in a table ***T*** stores all values of ***A*** in the same order as they appear in ***T***. Each row of the Projection Index stores one value of ***A***. The row order of value ***x*** in the index is the same as the row order of value ***x*** in ***T*** [12]. Projection Index on the columns reduces terrifically the cost of querying because a single I/O operation may bring more values into memory. Sybase has such an indexing technique named FastProjection Index on every column of a table. The main purpose of projection indexes is to reduce the cost of querying a particular attribute field. Projection indexes work faster than other techniques when only the column values are desired as opposed to the table rows themselves, because the actual tuples of the fact table need not be accessed at all.

Merit of projection indexing is:

- It speeds up the performance when a few columns in the table are retrieved.

Demerit of projection indexing is:

- It can be used only to retrieve raw data (i.e., column list in selection) [8].

IV. INDEXING COMPARITIVE STUDY

Indexing in data warehouse is very much essential these days, as importance of strategic information is increasing day by day. Companies need to analyze their data frequently and this can only be analyzed via querying on giant data warehouse. Querying on such a giant data warehouse, takes a long time to produce results of these queries. To solve this problem of long response, indexing is used. Indexing basically increases the speed of searching data depending upon different criteria's. Depending upon the size of data warehouse, cost, space requirement, types of queries to be run on etc., indexing technique is chosen. There are different indexing techniques possible, but according to the requirement of company, data warehouse design and expected queries to be run on (i.e. expected outcomes from the data warehouse) indexing technique is being chosen.

Table 1 shows the comparative study of five different indexing techniques, based upon some specific domains; depending upon which one can choose which technique they can apply to their data warehouse. Indexing type should to chosen such that it is cheap, uses less space, is more efficient in terms of query processing speed and most importantly produces correct results. Depending upon the type of queries to be run onto the data warehouse, indexing technique performance also changes. All techniques don't work for star support query, e.g. only bitmap join index works for star query. Projection index works well for queries when few columns are retrieved, similarly pure bitmap indexing works efficiently with columns involving scalar functions and b-tree indexing works efficiently for known queries and so on.

So, this paper provides a comparative study of few indexing techniques to solve a very big problem of choosing any one of these techniques. At least this paper will provide an idea to people that how to increase efficiency of query performance. Hence, it is basically provided to speed up data retrieval access speed.

Table 1: Indexing Comparative Study

Indexing Techniques / Properties	B-Tree Index	Pure Bitmap Index	Encoded Bitmap Index	Bitmap Join Index	Projection Index
Basic Characteristic	It is being implemented at the leaves of the index. It has 2 representations i.e. row id and bitmap; one is opted depending upon the cardinality of the data.	It uses equality encoding scheme. An array of bits is used to represent column value of each row in a table, where values are set according to the following convention: ON (1), OFF(0)	In this technique, indexes are basically bit-sliced index are built on attribute domain.	The index is built in the fact table but by the restriction of a column on the dimension table.	The actual values of the columns of the indexed table are stored to built index here.
Column Cardinality	Suited for high cardinality columns	Suited for low cardinality columns	Suited for low cardinality columns	Suited for low cardinality columns	Suited for high cardinality columns
Space Requirement	Space requirement is independent of cardinality of indexed columns	Uses very less space	Uses space efficiently	_____	_____
Cost	Relatively inexpensive when we update the indexed column	Very expensive when we update index column	Expensive as rebuilt every time when a new indexed value for which we run out of bit to represent is added.	Expensive as rebuilt every time	Relatively inexpensive
Speed of querying	Speeds up known queries	Performs efficiently with columns involving scalar functions	Performs efficiently with wide range query	_____	Speeds up the performance when a few columns in the table are retrieved
Indexes Combination possibility	The indexes cannot be combined before fetching the data	The indexes can be combined before fetching raw data	_____	_____	_____
Star Query Support	Does not support	Does not support	Does not support	Supports star queries	Does not support

V. CONCLUSION

This paper basically shows the difference between different indexing techniques by first showing the difference between bitmap indexing (pure bitmap, encoded bitmap, bit-slice), projection indexing, b-tree indexing upon criteria's like as basic characteristic, column cardinality, space requirement, cost, speed of querying, index combination support and star query support. Then, they are explained individually depending upon their merits and demerits. The main focus of this paper is on how and when to choose, which kind of indexing technique. The paper would give a basic idea to company's data warehouse designers that what they should do to optimize query performance i.e. use of indexing and which kind of indexing (depending upon the requirement).

REFERENCES

[1] Paulraj Ponniah, Data Warehousing Fundamentals, Wiley India Pvt. Ltd., Reprint 2008.

[2] Inmon, W.H., Building the Data Warehouse (2nd Edition), New York: Wiley, 1996.

[3] Transaction Processing Performance Council (TPC), "TPC Benchmark D, Decision Support", Standard Specification Revision 2.0.1, December 5, 1998, http://www.tpc.org.

[4] OLAP Council, "APB-1 OLAP Benchmark Release II", November 1998.http://www.olapcouncil.org

[5] Wikipedia, http://en.wikipedia.org.

[6] Vanichayobon S., Gruenwald L., "Indexing Techniques for Data Warehouses' Queries"

[7] Maher G. El, Haddouti H., "Bitmap indexing and related indexing techniques"

[8] Ghanjaoui Y., Haddouti H., "Indexing Techniques in Data Warehousing Environment the UB-Tree Algorithm"

[9] Weigel F. ,"A Survey of Indexing Techniques for Semistructured Documents"

[10]] Azefack S.,Aouiche K. andDarmont J., "Dynamic index selection in data warehouses

[11] P. O'Neil and G. Graefe, "Multi-Table joins through Bitmapped join indices", SIGMOD Record, Vol. 24, No. 3, Sep. 1995.

[12] P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes", SIGMOD, 1997.