Data Warehouse Tuning: The Supremacy of Bitmap Index

EL AMIN AOULAD ABDELOUARIT
Laboratory modeling and information theory
Abdelmalek Essaadi University, Tétouan, Morocco

Mohamed El Merouani
Laboratory modeling and information theory
Abdelmalek Essaadi University, Tétouan, Morocco

Abdellatif Medouri
Laboratory modeling and information theory
Abdelmalek Essaadi University, Tétouan, Morocco

ABSTRACT
The data query is the only way to get information from a data warehouse, for that, the designer should consider it effectiveness while the selection of relevant indexes and their combination with materialized views, also the index selection is known as a NP-complete problem, as per the number of indexes is exponential in the total attributes in the database, This makes the choose of the suitable index type is a necessary step while the data warehouse design.

In this paper we show that Bitmap index is more advantageous than the B-tree index, based on three factors, size of index, clustering factor and compression, with a real experiment.

General Terms
Business Intelligence, Data warehouse.

Keywords
Data Warehouse DBMS, indexes, business intelligence.

1. INTRODUCTION
Some administration tasks that are taken by a data warehouse administrator needs to be decided before the data warehouse implementation, like logical and physical database design, management of storage space and performance tuning.

Physical design are considered as the most important task, including data organization and improving access to these data, the fast access to this data requires to use general index to find the information wanted without reviewing all data [1], [3], [5], [7].

Index selection is difficult because their number is exponential in the total number of attributes in the database. So the index plays an important role in the performance of databases, for that we focus on this aspect of the data warehouse, which it considers the focus of the designer when editing and query optimization selection.

The objective is to minimize the query execution time. And as queries in a data warehouse are based on the index, we will work on the problem of choosing the type of index when designing our warehouse data.

Also to optimize storage usage, the index compression problem is an important point that need to be checked

There are several types of indexes supported by databases such as Bitmap [4] B-tree [3], [6], [7], [8], Bitmap join [9], range-based bitmap index [10] etc.. In this sense we have chosen two types of index relevant to this study, the index type: B-tree index and type Bitmap.

2. BITMAP INDEX
2.1 Definition
A bitmap index is a data structure defined in a DBMS used to optimize access to data in databases. It is a type of indexing is particularly interesting and effective in the context of selection queries. The index bitmap attribute is encoded in bits, where its low cost in terms of space occupied. [7] All possible attribute values are considered, the value is present or not in the table. Each of these values is an array of bits, called bitmap, which contains as many bits as n-tuples present in the table. Thus, this type of index is very effective when the attributes have a low number of distinct values. Each bit represents the value of an attribute for a given tuple. For each bit, there is an encoding presence / absence (1/0), which indicates that a tuple or not the present value characterized in bitmap.

Table 1: Basic Bitmap adopted by [9]

ROWID	C	B0	B1	B2	B3
0	2	0	0	1	0
1	1	0	1	0	0
2	3	0	0	0	1
3	0	1	0	0	0
4	3	0	0	0	1
5	1	0	1	0	0
6	0	1	0	0	0
7	0	1	0	0	0
8	2	0	0	1	0

To illustrate how a bitmap index works, we take an example EE-PP-O'Neil and O'Neil [2]. Table 1 illustrates a basic bitmap index into a table containing 9 records, where the index is created in the C column with integers ranging from 0 to 3, we say that the cardinality of the column C is 4, by what there are 4 distinct values [0, 1, 2, 3], where the index bitmap C Contains 4 bitmaps shown as B0, B1, B2 and B3 corresponding value represents. In this example, the first line where RowID = 0, column C is worth 2, consequently, B2 column bit value "1", while the other bitmaps are set to "0". Same for the next line, where C = 1 corresponds to the bitmap B1 is set to 1 and the rest to "0". This process is repeated for the remaining lines. [12]

2.2 Properties

Bitmap indexes have a very interesting property of responding to certain types of requests without returning the data themselves, thus optimizing the response time, disk storage. This is possible by counting operations (COUNT) and logical operators (AND, OR, etc.) that act "bit by bit" on bitmaps.

3. B-TREE INDEX

3.1 Definition

The index B-tree stores the index values and pointers to other index nodes using a recursive tree structure. [3], [6], [7], [8] The data are easily identified by traces pointers. The highest level of the index is called the root while the lowest level is called the leaf node or "leaf node". [7] All other levels between them are called branches or internal nodes. All roots and branches contain entries that point to the next level of the index. Leaf nodes consist of the index key and pointers pointing to the physical location of records. We present details of the index B-tree structure [7].

The B-tree structure is used by the database server to configure the index (Figure 1)

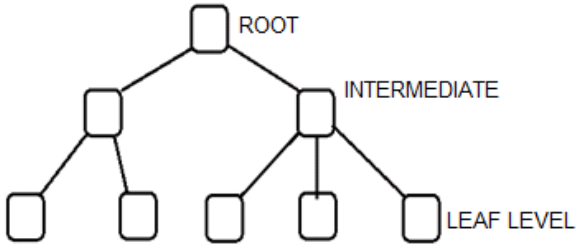

Fig 1: B-tree structure

Root or root is the highest level of the index points to the following levels of nodes branches.

Intermediate nodes or branches contain pointers to the following branches or to the leaf nodes level.

Node leaves or leaf nodes: the lowest level of the index points to other node leaves.

4. HYPOTHESIS

The conventional wisdom is that bitmap indexes are most appropriate for columns having low distinct values - such as gender, marital status, and relationship. This assumption is not entirely accurate, however. In reality, a bitmap index is always advisable for systems in which the data is not updated frequently by many competing systems. In fact, as I will demonstrate here, a bitmap index on a column with unique values to 100% (candidate of the primary key column) is as effective as a B-tree index.

In other side the compression is also an important point to check, I will try to compare

5. ANALYSIS AND RESULTS

As known, the bitmap index is more efficient than the b-tree index by its low cardinality columns, we present in this experimentation the performance given by Bitmap index comparing with the B-tree.

The factors that are used to make this comparison are:

- The index size
- Clustering factor
- Compression

5.1 Index size comparison

Step 1:

In our Data warehouse schema, we created a table named “Employees” with 100000 records and with a column named employee_id with 100000 distinct values and we added a GRADE column with 4 distinct values only.

Step 2:

We create now a standard B-Tree index on the GRADE column using this SQL :

SQL> create index employees_grade_i on employees(grade);

Then we check the index size using this query :

SQL> select index_name, index_type, distinct_keys, blevel, leaf_blocks from dba_indexes where index_name=’EMPLOYEES_GRADE_I’;

And we got this result:

Table 2: B-tree index size checking result in GRADE column

INDEX_ NAME	INDEX _TYPE	DISTINCT _KEYS	BLEVEL	LEAF_ BLOCKS
employees_grade_i	Normal	4	1	176

Step 3: Creating a bitmap index on the same column to compare the size (dropping the b-tree index created in first step)

SQL> create bitmap index employees_grade_bitmap_ii on employees(grade);

Step 4: In the bitmap index size checking, we use the same query:

SQL> select index_name, index_type, distinct_keys, blevel, leaf_blocks from dba_indexes where index_name=’EMPLOYEES_GRADE_II’;

Table 3: Bitmap index size checking result in GRADE column

INDEX_ NAME	INDEX _TYPE	DISTINCT _KEYS	BLEVEL	LEAF_ BLOCKS
employees_grade_ii	Bitmap	4	1	10

Note that the index size is reduced from 176 to 10 (while going from B-tree to bitmap index)

Step 5: the bitmap index creation on employee_id column that contains 100000 distinct values:

SQL> create bitmap index employees_empid_bitmap_i on employees(employee_id).

By checking the index size using the same query, we have this table as result:

Table 4: Bitmap index size checking result in EMPLOYEE_ID column

INDEX_ NAME	INDEX _TYPE	DISTINCT _KEYS	BLEVEL	LEAF_ BLOCKS
employees_empid_ bitmap_i	Bitmap	100000	1	348

And when trying with B-tree index we have this result:

Table 5: B-Tree index size checking result in EMPLOYEE_ID column

INDEX_ NAME	INDEX _TYPE	DISTINCT _KEYS	BLEVEL	LEAF_ BLOCKS
employees_empid_ btree_i	B-tree	100000	1	222

5.2 Clustering factor checking

For large distinct values B-tree index occupies less size, and for minimal distinct values, the bitmap index occupies less size.

Clustering factor: considered as the sum of rows orders in a table based on the index values.

- If this amount is near the number of blocks, then the table order is well done, and the index entries in a single leaf block are pointing to rows stored in the same data blocks.
- If the value is near the number of rows, then the table is randomly ordered, so is improbable that the index entries in a single leaf block are pointing to rows stored in the same data blocks.

Table 6: Clustering factor and blocks used for B-Tree index on GRADE column

INDEX_NAME	CLUSTERING _FACTOR	BLOCKS
employees_grade_i	1148	256

Table 7: Clustering factor and blocks used for Bitmap index on GRADE column

INDEX_NAME	CLUSTERING _FACTOR	BLOCKS
employees_grade_ii	20	16

5.3 Compression advantage:

Is known that bitmap Indexes store a separate bitmap for every distinct value of a column, so it's like the storage space would become prohibitive for large tables with a lot of distinct values. As example, if a table contains 1 million rows with 10,000 distinct values, won't the Bitmap Index contain 1,000,000 x 10,000 = 10,000,000,000 bits near 1.2GB. A million-row table is considered small for a data warehouse, but 1.2GB is very big.

As Oracle uses some clever compression where long sequences of 1s or 0s in the bitmap consume hardly any space, storage space is not a significant consideration for bitmap indexes with a large number of distinct values.

With a lot of distinct values, instances of a particular value will be widely spread. This means long strings of zeros in the bitmap which become highly compressed.

The effect illustration is better shown with an example. First, create a table with:

- 1 million rows
- 2 identical columns with 10,000 distinct values
- A B-Tree index on one column
- A Bitmap Index on the other column

Table 8: Table comp_test creation query

```
SQL> CREATE TABLE comp_test
  2  AS
  3  SELECT mod(LEVEL, 10000) AS col1
  4  ,  mod(LEVEL, 10000) AS col2
  5  FROM   dual
  6  CONNECT BY LEVEL <= 1000000;
Table created.
SQL> CREATE INDEX COMP_TEST_col1_ix ON
COMP_TEST(col1);
Index created.
SQL> CREATE BITMAP INDEX
COMP_TEST_col2_bx ON COMP_TEST(col2);
Index created.
```

Now, compare the size of each index.

Table 9: Index size comparison

```
SQL> SELECT segment_name,
sum(bytes)/1024/1024 AS mb
  2  FROM   user_segments
  3  WHERE  segment_name LIKE 'COMP%'
  4  GROUP BY segment_name;

SEGMENT_NAME                         MB
------------------------------ --------
COMP_TEST                            15
COMP_TEST_COL2_BX                     4
COMP_TEST_COL1_IX                    17
```

Note that the Bitmap Index has been squished down to just 4MB – one-quarter the size of the B-Tree index! In fact, I ran this same test with 1 million distinct values (ie. One row per value) and still the bitmap index was only 50% larger than the B-Tree index.

5.3.1 Results

The most difficult scenario of bitmaps compression si when it contains strings of zeros that are interrupted by one or two "ones" in each byte. It won't compress at all, but if every 8th row has the same value then there can only be 8 such uncompressible bitmaps for the column.

In other words:

There is many distinct values means each bitmap will be very compressed because they contain a lot of zeros.

The bitmaps are not very well compressed then there will not be many of them because there are only a few distinct values

6. CONCLUSION AND FUTURE WORK

The optimizer did a full scan to a table when using the B-tree index, this caused a higher clustering factor, whereas in the case of bitmap indexes the clustering factor is too low comparing with the first one. Here we deduct the performance by the sum of input/output required to fetch the result.

The message here is well clear. Both indexes have a similar goal: to return results as fast as possible. But the choice of which one to use should depend only on the type of application, and not on the level of cardinal.

In other side, the compression cannot be a factor that affects our choice of Bitmap index for data warehouse optimization strategy.

As future work, another study will be done on data warehouse schema comparison, specially it impact on data warehouse performance.

7. REFERENCES

[1] S. Chaudhuri, U. Dayal,An Overview of Data Warehousing and OLAP Technology., ACM SIGMOD RECORD. 1997

[2] E. E-O'Neil and P. P-O'Neil, Bitmap index design choices and their performance impli-cations, Database Engineering and Applications Symposium. IDEAS 2007. 11th International, pp. 72-84.

[3] R. Kimball, L. Reeves, M. Ross, The Data Warehouse Toolkit. John Wiley Sons, NEW YORK, 2nd edition, 2002

[4] W. Inmon, Building the Data Warehouse., John Wiley Sons, fourth edition, 2005

[5] C. DELLAQUILA and E. LEFONS and F. TANGORRA, Design and Implementation of a National Data Warehouse. Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 pp. 342-347

[6] D. Comer,Ubiquitous b-tree, ACM Comput. Surv. 11, 2, 1979, pp. 121-13

[7] R. Strohm, Oracle Database Concepts 1g,Oracle, Redwood City,CA 94065, 2007

[8] C. Dell aquila and E. Lefons and F. Tangorra, Analytic Use of Bitmap Indices. Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, CorfIsland, Greece, February 16-19, 2007 pp. 159

[9] P. O'Neil and G. Graefe, Multi-table joins through bitmapped join indices, ACM SIGMOD Record 24 number 3, Sep 1995 , pp. 8-11.

[10] K. Wu and P. Yu, Range-based bitmap Indexing for high cardinality attributes with skew, In COMPSAC 98: Proceedings of the 22nd International Computer Software and Applications Conference. IEEE Computer Society, Washington, DC, USA, 1998, pp. 61-67.