Indexes in Data Warehouse

HADIS GHAFOURI

Indexes

- It is a data structure technique which is used to quickly locate and access the data in a database.
- a way to optimize the performance.
- minimizing the number of disk accesses.
- too few indexes, the data loads quickly but the query response is slow.
- too many indexes, the data loads slowly and Increase storage requirements but the query response is good.
- reduces the time to see query results.(large tables and complex queries)

Criteria for selecting indexes

1) type of data warehouse you have.(archive or real-time)
2) how large the dimensions and fact tables are.(partitioned fact tables)
3) who will be accessing the data and how they'll do.(ad hoc or structured application interfaces)
4) Number of unique values
5) Data types

Bitmap Indexes

- A **bitmap index** is a special kind of database index that uses bitmaps.
- work well for *low-cardinality columns*, which have a modest number of distinct values.
- The extreme case of low cardinality is Boolean data(True and False).
- Bitmap indexes use bit arrays (commonly called bitmaps).
- answer queries by performing bitwise logical operations on bitmaps.
- significant space and performance advantage over other structures for query of such data.

Bitmap Indexes

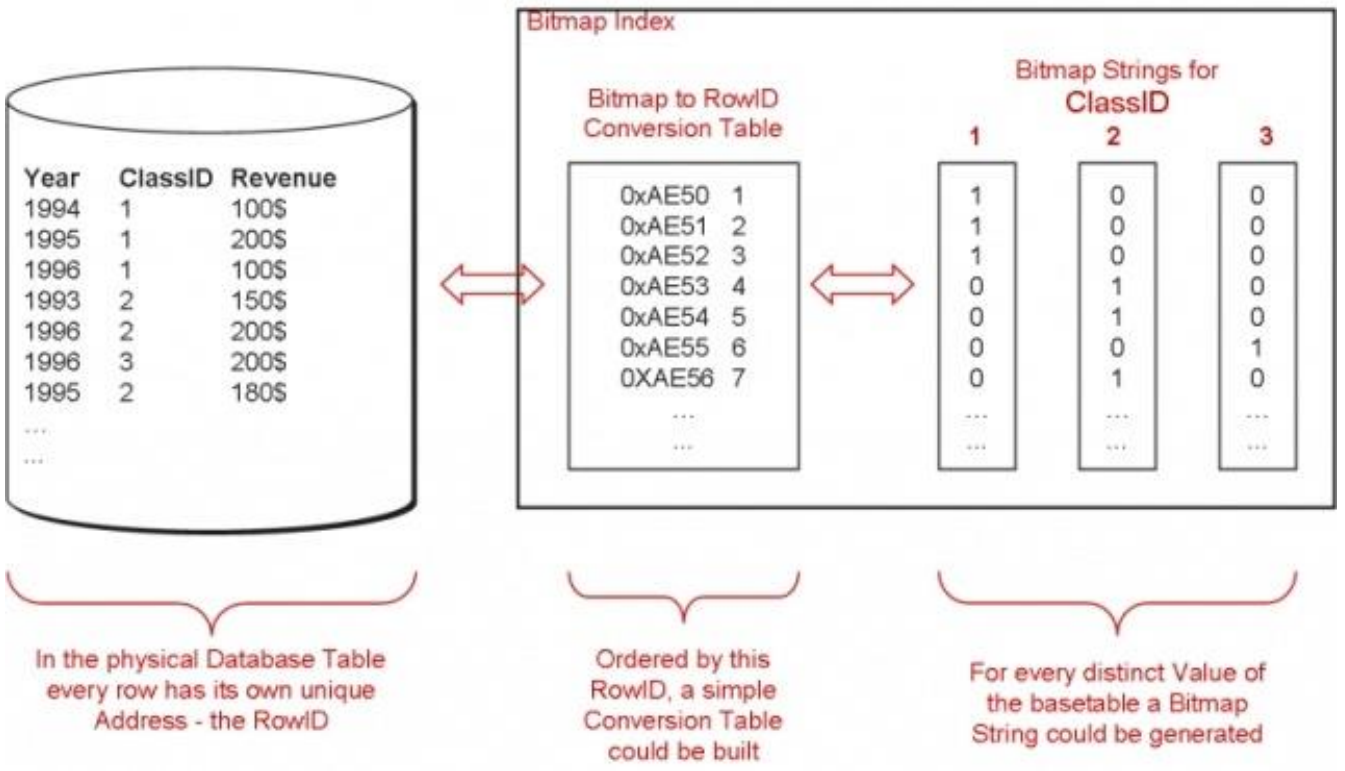
- Drawback: less efficient than the traditional B-tree indexes for columns whose data is frequently updated
- employed in read-only systems for fast query.(data warehouses)
- unsuitable for online transaction processing applications.
- primarily intended for data warehousing applications where users query the data rather than update it.
- They are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

Benefits for Data Warehousing Applications

- 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.
- Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of disk space because the indexes can be several times larger than the data in the table.
- Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

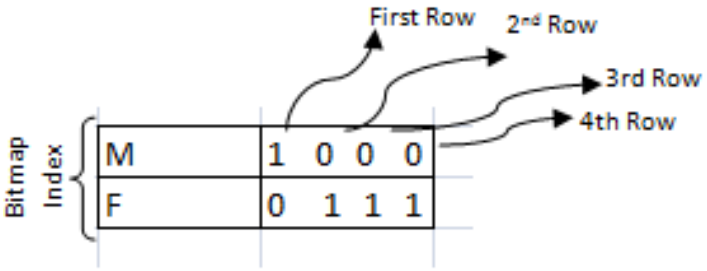
Bitmap Indexes

- strings of 0s and 1s for each distinct value.
- records with *class_id* = 2 : row 4,5 and 7
- where *class_id* = 1 or 2 : OR together the two strings.
- efficiently combine multiple single column indexes into one on execution time.



Bitmap Indexes

- An index provides pointers to the rows in a table that contain a given key value.
- A regular index stores a list of rowids for each key corresponding to the rows with that key value.
- In a bitmap index, a bitmap for each key value replaces a list of rowids.
- Each bit in the bitmap corresponds to a possible rowid, and if the bit is set, it means that the row with the corresponding rowid contains the key value.
- mapping function: converts the bit position to an actual rowid(same functionality as a regular index).
- Bitmap indexes store the bitmaps in a compressed way. (Small #of distinct key values , better compress to a B-tree index)
- a bitmap index works best if the number of distinct values is less than 0.1% of the total row count.
- So for a million row table, all columns with less than 1000 distinct values are perfect candidates.



Bitmap Indexes

- **Need of Bitmap Indexing**
- company holds an employee table
- entries like EmpNo, EmpName, Job, New_Emp and salary.
- employees are hired once in the year(updated very less)
- columns will be frequently used in queries to retrieve data.(No. of female employees in the company)
- need a file organization method(fast enough to give quick results)
- traditional file organization method is not that fast

EmpNo	EmpName	Job	New_Emp	Salary
1	Alice	Analyst	Yes	15000
2	Joe	Salesperson	No	10000
3	Katy	Clerk	No	12000
4	Annie	Manager	Yes	25000

How Bitmap Indexing is done

- column *New_Emp* has only two values **Yes** and **No**.
- Job of the Employees is divided into 4 categories (Manager, Analyst, Clerk and Salesman).
- columns with low cardinality.(less unique values, they can be queried very often)
- **Bit:** 0/1or true/false or yes/no.
- bits are used to represent the unique values in those low cardinality columns.
- This technique of storing the low cardinality rows in form of bits are called bitmap indices.

EmpNo	EmpName	Job	New_Emp	Salary
1	Alice	Analyst	Yes	15000
2	Joe	Salesperson	No	10000
3	Katy	Clerk	No	12000
4	Annie	Manager	Yes	25000

Bitmap Example

- If *New_Emp* is the data to be indexed, the content of the bitmap index is shown as four(As we have four rows in the above table) columns under the heading Bitmap Indices.
- Here Bitmap Index "Yes" has value 1001 because row 1 and row four has value "Yes" in column *New_Emp*.
- In this case there are two such bitmaps, one for "New_Emp" Yes and one for "New_Emp" NO.
- each bit in bitmap indices shows that whether a particular row refer to a person who is New to the company or not.

EmpNo	EmpName	Job	New_Emp	Salary
1	Alice	Analyst	Yes	15000
2	Joe	Salesperson	No	10000
3	Katy	Clerk	No	12000
4	Annie	Manager	Yes	25000

New_Emp Values	Bitmap Indices
Yes	1001
No	0110

Bitmap Example

- Most columns will have more distinct values. For example the column Job here will have only 4 unique values.
- find out the details for the Employee who is not new in the company and is a sales person query:

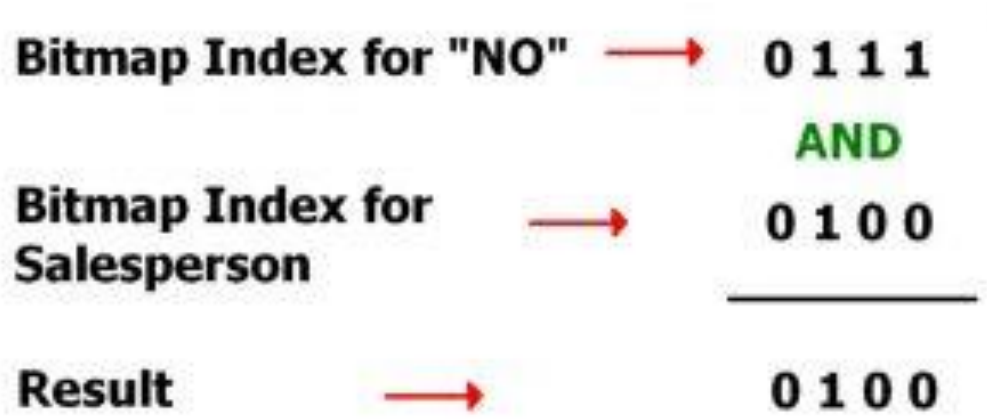
```
SELECT *
        FROM Employee
            WHERE New_Emp = "No" and Job = "Salesperson";
```

Job Values	Bitmap Indices
Analyst	1000
Salesperson	0100
Clerk	0010
Manager	0001

Job column the bitmap Indexing

Bitmap Example

- For this query the DBMS will search the bitmap index of both the columns and perform logical AND operation on those bits and find out the actual result:



- Here the result 0100 represents that the second row has to be retrieved as a result.

Bitmap Indexing in SQL

- The syntax for creating bitmap index in sql is given below:

```
CREATE BITMAP INDEX Index_Name
         ON Table_Name (Column_Name);
```

- For the above example of employee table, the bitmap index on column New_Emp will be created as follows:

```
CREATE BITMAP INDEX index_New_Emp
        ON Employee (New_Emp);
```

Bitmap Indexes Usage

- Bitmap indexes are most effective for queries that contain multiple conditions in the WHERE clause.
- If you are unsure of which indexes to create, the SQL Access Advisor can generate recommendations on what to create.
- As the bitmaps from bitmap indexes can be combined quickly, it is usually best to use single-column bitmap indexes.
- When creating bitmap indexes, you should use NOLOGGING and COMPUTE STATISTICS.(Oracle)

Bitmap Index Benefits

- Work with Parallel query and parallel DML.
- supports parallel create indexes and concatenated indexes.
- Betst for columns in which the ratio of the number of distinct values to the number of rows in the table is small. We refer to this ratio as the **degree of cardinality**.
- A gender column, which has only two distinct values (male and female), is optimal for a bitmap index.
- However, data warehouse administrators also build bitmap indexes on columns with higher cardinalities.
- B-tree indexes are most effective for high-cardinality data: that is, for data with many possible values, such as customer_name or phone_number.

Candidates for Using a Bitmap Index

- the fact table is queried alone or when the fact table is joined with two or more dimension tables, and there are indexes on foreign key columns in the fact table.
- A fact table column is a candidate for a bitmap index when :
  1. There are 100 or more rows for each distinct value in the indexed column. When this limit is met, the bitmap index will be much smaller than a regular index, and you will be able to create the index much faster than a regular index. An example would be one million distinct values in a multi-billion row table.
  2. The indexed column will be restricted in queries (referenced in the WHERE clause).
  3. The indexed column is a foreign key for a dimension table.

Bitmap Indexes and Nulls

- Unlike most other types of indexes, bitmap indexes include rows that have NULL values.
- Indexing of nulls can be useful for some types of SQL statements(aggregate function COUNT).
- ***Example Bitmap Index***

```
SELECT COUNT(*) FROM customers WHERE cust_marital_status IS NULL;
```

- bitmap index on *cust_marital_status*.
- this query would not be able to use a B-tree index, because B-tree indexes do not store the NULL values.

Advantages, Disadvantages

- **Advantages of Bitmap Indices**
- faster retrieval of the records when there are less cardinality columns and those columns are most frequently used in the query. This method is efficient even if we have very big table.
- This method is more efficient when the columns have least involved in insert/update/delete operations.
- allows to combine multiple bitmap indices together.

- **Disadvantages of Bitmap Indices**
- not suitable for small tables. In small tables, DBMS will force to use full table scan instead of using bitmap index.
- cause deadlock When there is multiple insert/update/delete on the table from different users. It will take time to perform the DML transaction and then to update the bitmap index.
- When there is multiple DML transaction from different users, it will not be able to perform transaction quickly, and causing the deadlock.
- When there is large number of records, there is an overhead to maintain this bitmap indexes.
- When new record is entered, we have to modify the bitmap index throughout, which is a tedious and time consuming.

Refrences

- https://docs.oracle.com/cd/E11882_01/server.112/e25554/indexes.htm#DWHSG8139
- https://www.geeksforgeeks.org/bitmap-indexing-in-dbms/
- https://wiki.scn.sap.com/wiki/display/EIM/Bitmap+Indexes