Indexing Techniques In Data Warehouses

Introduction

Most of the queries against a large data warehouse are complex and iterative, and the ability to answer these queries efficiently is a critical issue in the data warehouse environment. So, if the right index structures are built on columns, the performance of queries, especially ad-hoc queries will be greatly enhanced.

A column has its own characteristics which we can use to choose a proper index. These characteristics are given in the following:

- *Cardinality data* which is the number of distinct values in the column.
- *Distribution* which is the occurrence frequency of each distinct value of the column.
- *Value range*

The characteristics of indexing that we have to concern

- The index should be small and utilize space efficiently
- The index should be able to operate with other indexes to filter out the records before accessing raw data.
- The index should support ad hoc and complex queries and speed up join operations.
- The index should be easy to build (easily dynamically generate), implement and maintain.

Indexing Techniques in Data Warehouses

In data warehouse systems, there are many indexing techniques. Each existing indexing technique is suitable for a particular situation. So, here we will discuss the various types of indexes which is commonly used in data warehouse.

1. The B-Tree Index

The B-Tree Index is the default index for most relational database systems. popular in data warehouse applications for high cardinality column such as names since the space usage of the index is independent of the column cardinality. However, this type of index has some c/cs which it a poor choice for DWH's quires. Theses c/cs likes, in case of low cardinality data such as the gender column since it reduces very few numbers of I/Os and may use more space than the raw indexed column. Also, it fetches the result data ordered by the key values which have unordered row ids, so more I/O operations and page faults are generated.

2. Bitmap Index

The bitmap representation is an alternate method of the row ids representation. It is simple to represent, and uses less space- and CPU-efficient than row ids when the number of distinct values of the indexed column is low.

These types of indexes improve complex query performance by applying low-cost Boolean operations such as OR, AND, and NOT in the selection predicate on multiple indexes at one time to reduce search space before going to the primary source data.

There are Many variations of the Bitmap Index shush as (Pure Bitmap Index, Encoded Bitmap.

- Pure Bitmap

  It consists of a collect of bitmap vectors each of which is created to represent each distinct value of the indexed column. E.g. a bit *i* in a bitmap vector, representing value *x*, is set to *1* if the record *i* in the indexed table contains *x*.

  The bitmap vectors of the values specified in the predicate condition are read into memory. If there are more than one bitmap vectors read, a Boolean operation will be performed on them before accessing data. However, the sparsity problem occurs if the Pure Index is built on high cardinality column which then requires more space and query processing time to build and answer a query.

- Encoded Bitmap

  The lookup table stores the mapping between A and its encoded representation Comparing with the Pure Bitmap Index, the Encoded Bitmap Index improves the space utilization, and solves sparsity problems. The size of it built on the high cardinality column is less than the Pure Bitmap Index.

3. Join Index

This type of 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, depending on the cardinality of the indexed column. It is called Bitmap Join Index, is used with the low cardinality data while a row id representation is used with a high cardinality.

4. Projection Index

This type of index is index is built by storing actual values of column(s) of indexed table. It speeds up the performance when a few columns in the table are retrieved.

References

S. Vanichayobon and L.Gruenwald, “Indexing Techniques for Data Warehouses’ Queries”, Norman,73019.