Information Technology Institute
Data Management Track
Data Warehouse Project (Intake42)

Indexes in Data warehouse

Bitmap indexes

Bitmap indexes are widely used in data warehousing environments. The environments typically have large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. For such applications, bitmap indexing provides:

- 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.

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. A mapping function converts the bit position to an actual rowid, so that the bitmap index provides the same functionality as a regular index.

B-Tree Indexes

A B-tree index is organized like an upside-down tree. The bottom level of the index holds the actual data values and pointers to the corresponding rows, much as the index in a book has a page number associated with each index entry.

In general, use B-tree indexes when you know that your typical 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.

B-tree indexes are most commonly used in a data warehouse to index unique or near-unique keys. In many cases, it may not be necessary to index these columns in a data warehouse, because unique constraints can be maintained without an index, and because typical data warehouse queries may not work better with such indexes. B-tree indexes are more common in environments using third normal form schemas. In general, bitmap indexes should be more common than B-tree indexes in most data warehouse environments.

Partitions in Data warehouse

Hash Partitioning

Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to a partitioning key that you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size. Hash partitioning is the ideal method for distributing data evenly across devices. Hash partitioning is also an easy-to-use alternative to range partitioning, especially when the data to be partitioned is not historical.

Oracle Database uses a linear hashing algorithm and to prevent data from clustering within specific partitions, you should define the number of partitions by a power of two (for example, 2, 4, 8).

The following statement creates a table sales_hash, which is hash partitioned on the salesman_id field:

```
CREATE TABLE sales_hash
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_amount  NUMBER(10),
week_no       NUMBER(2))
PARTITION BY HASH(salesman_id)
PARTITIONS 4;
```

List Partitioning

List partitioning enables you to explicitly control how rows map to partitions. You do this by specifying a list of discrete values for the partitioning column in the description for each partition. This is different from range partitioning, where a range of values is associated with a partition and with hash partitioning, where you have no control of the row-to-partition mapping. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way. The following example creates a list partitioned table grouping states according to their sales regions:

```
CREATE TABLE sales_list
(salesman_id  NUMBER(5),
salesman_name VARCHAR2(30),
sales_state   VARCHAR2(20),
sales_amount  NUMBER(10),
sales_date    DATE)
PARTITION BY LIST(sales_state)
(PARTITION sales_west VALUES('California', 'Hawaii') COMPRESS,
 PARTITION sales_east VALUES('New York', 'Virginia', 'Florida'),
 PARTITION sales_central VALUES('Texas', 'Illinois'));
```