Database Indexing

An index on a table is a mechanism which provides direct access to the physical disk block location of table records based on the values contained in certain indexed columns. An analogy can be made with the index in the back of a book – it tells exactly what pages to go to in order to find something.

Using the index to locate disk block containing the data is much quicker than linearly searching.

The index is maintained as a separate data structure that uses the indexed field value as a lookup key. The index then returns the physical location where records holding that key value are stored. The use of an index can greatly improve the performance of SQL Select statements where the fields used to Join tables and the fields used in Where clauses are set up as indexes.

An index is 'sparse' if there are very few records matching a specific key value. A 'dense' index has lots of records matching a key value. Indexes give the greatest performance benefit when used on sparse fields.

An index provides a performance boost when reading data from a database. However, there is a performance penalty for database update operations. This is because the index of table must also be updated each time a new row is added or an indexed field is updated. However, some Update operations include a Where clause to select a subset of records to update. This is essentially a READ (before UPDATE) operation and therefore might benefit from an index i.e. the read benefit might outweigh the update hit.

Database admins constantly monitor the performance of the database under different load conditions and revise their use of indexes accordingly.

­Index Structures

(Use in Assignment for Optimization)

There are many different ways to store an index internally in the database. An index should be quick to search and also update. Different index types exist to suit different types of databases, i.e. depending on the frequency of index keys etc. Popular types are the hash table index and the B-Tree index.

Hash Table

The hash table index applies an algorithm to the key value to give the physical block no.

Index value 🡪 hash function 🡪 Database Block No.

B- Tree Index (B+ - Tree)

The B-Tree Index is a commonly used method of storing index data. The indices are organised as a balanced tree, similar to a binary tree. Entries are divided based on their value e.g. an integer – values less than 20 are in one branch of the tree, 20-50 is in another branch, 80+ is in another branch etc. Each branch may have more branches and so on. So, based on the value being searched for, the read command will traverse the tree quickly to the relevant branch and return the value.

B-Tree indexes are relatively easy to update when a new record is inserted or a record is deleted/updated.

IN regular hash tables, many objects that generate the same hash value will cause some cells to be overfilled i.e. need to be searched and defeat the point of indexing.

An index on column Y in table X can be created like this:

CREATE INDEX <indexname> as X(Y);

e.g. to create an index on 'county' in the User table:

CREATE INDEX county\_index AS User(County);

To delete the index use:

DROP INDEX county\_index;

Indexes can be made using more than one field:

CREATE INDEX county\_gender AS User(County, Gender);

Index are useful to use on foreign keys.