Why you would need a database index?

Let us understand this through a very simple example.

Suppose that we have a database table called Employee with three columns – Employee\_Name, Employee\_Age, and Employee\_Address. Assume that the Employee table has thousands of rows.

query to find all the details of any employees who are named ‘Jesus’?

SELECT \* FROM Employee

WHERE Employee\_Name = 'Jesus'

What would happen without an index on the table?

the database software would literally have to look at every single row in the Employee table to see if the Employee\_Name for that row is ‘Jesus’.

And, we can not just stop looking once we find just one row with the name ‘Jesus’, because there could be other rows with the name Jesus.

So, every row up until the last row must be searched – which means thousands of rows in this scenario will have to be examined by the database to find the rows with the name ‘Jesus’.

This is what is called a full table scan.

How a database index can help performance

The whole point of having an index is to speed up search queries by essentially cutting down the number of records/rows in a table that need to be examined.

What is an index?

An index is a data structure (most commonly a B- tree) that stores the values for a specific column in a table.

An index is created on a column of a table.

So, the key points to remember are that an index consists of column values from one table, and that those values are stored in a data structure.

What kind of data structure is an index?

B- trees are the most commonly used data structures for indexes.

The reason B- trees are the most popular data structure for indexes is due to the fact that they are time efficient – because look-ups, deletions, and insertions can all be done in logarithmic time.

And, another major reason B- trees are more commonly used is because the data that is stored inside the B- tree can be sorted.

The RDBMS typically determines which data structure is actually used for an index. But, in some scenarios with certain RDBMS’s, you can actually specify which data structure you want your database to use when you create the index itself.

How does a hash table index work?

hash tables are extremely efficient when it comes to just looking up values.

So, queries that compare for equality to a string can retrieve values very fast if they use a hash index.

For instance, the query we discussed earlier (SELECT \* FROM Employee WHERE Employee\_Name = ‘Jesus’) could benefit from a hash index created on the Employee\_Name column.

The way a hash index would work is that the column value will be the key into the hash table and the actual value mapped to that key would just be a pointer to the row data in the table.

Since a hash table is basically an associative array, a typical entry would look something like “Jesus => 0x28939?, where 0x28939 is a reference to the table row where Jesus is stored in memory.

Looking up a value like “Jesus” in a hash table index and getting back a reference to the row in memory is obviously a lot faster than scanning the table to find all the rows with a value of “Jesus” in the Employee\_Name column.

The disadvantages of a hash index

Hash tables are not sorted data structures, and there are many types of queries which hash indexes can not even help with. For instance, suppose you want to find out all of the employees who are less than 40 years old. How could you do that with a hash table index? Well, it’s not possible because a hash table is only good for looking up key value pairs – which means queries that check for equality (like “WHERE name = ‘Jesus'”). What is implied in the key value mapping in a hash table is the concept that the keys of a hash table are not sorted or stored in any particular order. This is why hash indexes are usually not the default type of data structure used by database indexes – because they aren’t as flexible as B- trees when used as the index data structure. Also see: Binary trees versus Hash Tables.

What are some other types of indexes?

Indexes that use a R- tree data structure are commonly used to help with spatial problems. For instance, a query like “Find all of the Starbucks within 2 kilometers of me” would be the type of query that could show enhanced performance if the database table uses a R- tree index.

Another type of index is a bitmap index, which work well on columns that contain Boolean values (like true and false), but many instances of those values – basically columns with low selectivity.

How does an index improve performance?

Because an index is basically a data structure that is used to store column values, looking up those values becomes much faster. And, if an index is using the most commonly used data structure type – a B- tree – then the data structure is also sorted. Having the column values be sorted can be a major performance enhancement.

What exactly is inside a database index?

Database index is created on a column in a table, and the index stores the values in that specific column. Database indexes also store pointers to the corresponding rows in the table. A pointer is just a reference to a place in memory where the row data is stored on disk.

This means that one of the values (or nodes) in the index for an Employee\_Name could be something like (“Jesus”, 0x82829), where 0x82829 is the address on disk (the pointer) where the row data for “Jesus” is stored

How does a database know when to use an index?

When a query like “SELECT \* FROM Employee WHERE Employee\_Name = ‘Jesus’ ” is run, the database will check to see if there is an index on the column(s) being queried. Assuming the Employee\_Name column does have an index created on it, the database will have to decide whether it actually makes sense to use the index to find the values being searched – because there are some scenarios where it is actually less efficient to use the database index, and more efficient just to scan the entire table. Read this article to understand more about those scenarios: Selectivity in SQL.

Can you force the database to use an index on a query?

Generally, you will not tell the database when to actually use an index – that decision will be made by the database itself. Although it is worth noting that in most databases (like Oracle and MySQL), you can actually specify that you want the index to be used.

How to create an index in SQL:

* Creating an index on the Employee\_Name column

CREATE INDEX name\_index

ON Employee (Employee\_Name)

* Creating a multi-column index in SQL:

CREATE INDEX name\_index

ON Employee (Employee\_Name, Employee\_Age)

What is a good analogy for a database index?

A very good analogy is to think of a database index as an index in a book. If you have a book about dogs and you are looking for the section on Golden Retrievers, then why would you flip through the entire book – which is the equivalent of a full table scan in database terminology – when you can just go to the index at the back of the book, which will tell you the exact pages where you can find information on Golden Retrievers. Similarly, as a book index contains a page number, a database index contains a pointer to the row containing the value that you are searching for in your SQL.

What is the cost/disadvantage of having a database index?

Well, for one thing it takes up space – and the larger your table, the larger your index. Another performance hit with indexes is the fact that whenever you add, delete, or update rows in the corresponding table, the same operations will have to be done to your index. Remember that an index needs to contain the same up to the minute data as whatever is in the table column(s) that the index covers.

As a general rule, an index should only be created on a table if the data in the indexed column will be queried frequently.