Index Overview

by Sophia

WHAT'S COVERED

This tutorial explores the different types of indexes in databases in two parts:

1. Indexes in General
2. Data Structures

1. Indexes in General

A database index is a structure that allows a query to efficiently retrieve data from a database. Let us look at a query run on our track table:

```
SELECT *
FROM track
WHERE album_id = 5;
```

If there are no indexes added to the table on this column, the database would have to scan row by row through the entire table to find all of the matching rows. If there were many rows in this table and we only expect to have a few rows returned, this can be quite an inefficient search. Imagine if there were one million rows in the table, and this album_id did not even exist. This means that the database would have to search through all one million rows to identify that there were no matching values at all. If the database has an index on the album_id column, however, the database would be able to much more efficiently find the matching data.

The concept of database indexes is quite similar to the alphabetical index at the end of a book. With a book, an individual can scan through the index fairly quickly to find the topics they're interested in and what page to turn to. This is a much faster approach than having to read through the entire book to find the content you want. The usefulness of a book index depends on the person who creates the topic list. Similarly, the database developer must determine what indexes could be useful.

Once an index is created in a database, nothing else needs to occur. The database will automatically update the index when the table data is modified. The database will also make use of the index if it will be more useful than searching row by row. Indexes can be useful not only in SELECT statements with joins on the indexed columns, but also in UPDATE and DELETE commands that have filtering criteria.

It is important to note that we do not want to index every single column in the database, as creating an index can take a long time. There is also a cost to maintaining the index. This depends on the database, but it could occur with every insert, update or delete SQL statement that affects the index. The index has to be

synchronized with the table, so if there are indexes that are not frequently used, they should ideally be removed. By default, primary keys and columns with the UNIQUE constraint have indexes created for them.

2. Data Structures

Most databases will implement indexes using one of the following data structures:

1. Hash index – this type of index is based on an order list of hash values. Typically we will have a hash algorithm that is used to create a hash value from a key column. This value then points to an entry in a hash table which then points to an actual location of the data row. Hash indexes can only handle simple equality operators and in most cases will be the first index to use when we use the equality operator = .
2. B-tree index – this type of index has the data organized in an upside-down tree. The index tree is stored separately from the data itself. The B-tree index self-balances, meaning that it will take about the same amount of time to access any row in the index regardless of the size of the index. For example, with one million rows of data, it will take less than 20 searches to find data using a B-tree index compared to one million searches if it were done sequentially. This is the most common type of index in databases and generally is the most useful when we have limited repeating values. B-trees can handle equality and range queries quite well. If the query uses a <, <=, =, >= or > operator, a B-tree index if available could be used. It is generally also used for pattern matching (using the LIKE clause) as long as the pattern is constant and starts with the beginning of the string being set.
3. Bitmap index – this type of index uses a bit array of zeros and ones. These are useful to represent a value or condition that may be true or false. For example, this could be useful to check if an individual was signed up for a newsletter. Typically, since the values in the columns that have the bitmap index are one of two values, the equality operator = is most commonly used.
4. Generalized Search Tree (GiST) – This is a more complex type of index unique to PostgreSQL that is not used often but focuses on certain functionality like searching for a value closest to another value or finding pattern matching. As such, there are special types of operators that could be useful for this purpose like <<, &< &>, >>, <<|, &<|, |&>, |>> @<, @>, ~= and &&. We have not explored any of these operators, because they are not used that frequently.
5. Generalized Inverted Index (GIN) – This is a unique type of index for PostgreSQL that focuses on indexing array values and testing if a value exists. Operators you would see with this include <@, @>, = and &&.

SUMMARY

Indexes are used to speed up queries and avoid having to search through data one row at a time.

Source: Authored by Vincent Tran