B-Tree Index

by Sophia Tutorial

WHAT'S COVERED

This tutorial explores the use of B-tree indexes in databases in three parts:

1. Introduction
2. Examples
3. When To Use

1. Introduction

As we discussed in the prior tutorial, the B-tree index is formatted like an upside-down tree. B-tree indexes generally handle equality and range values that could be sorted in a specific order. The most common operators that are used with a B-tree index include <, <=, =, >= and >. Note that there are constructs that are the same as these operators, like those that use BETWEEN or IN, as they could be represented in a similar manner. It is also common to see IS NULL or IS NOT NULL using the B-tree index as a means to check the data.

---

2. Examples

The database may also use a B-tree index with certain pattern matching operators such as LIKE, if the pattern is a constant and is anchored at the start of the string. For example, we could look for patterns of the name that starts with 'Wal':

```
SELECT *
FROM track
WHERE name LIKE 'Wal%';
```

Or for customers that have an email address starting with 'ro':

```
SELECT *
FROM customer
WHERE email LIKE 'ro%';
```

However, the B-tree index would not be useful if we tried to find information in the middle or at the end of the string, like tracks that have 'at' in the middle name:

```
SELECT *
FROM track
WHERE name LIKE '%at%';
```

Or customers that have the email with 'gmail.com' as the domain name:

```
SELECT *
FROM customer
WHERE email LIKE '%@gmail.com';
```

Other queries on data are based on ranges. For example, you could have open-ended ranges:

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

Or those that have specific ranges that contain values:

```
SELECT *
FROM track
WHERE album_id >= 5 AND album_id <=10;
```

This is the same as if we had:

```
SELECT *
FROM track
WHERE album_id BETWEEN 5 AND 10;
```

Note that this is different than considering two ranges that do not overlap like:

```
SELECT *
FROM track
WHERE album_id <= 5 AND album_id >=10;
```

Here we are looking for items with the album_id less than or equal to 5 while at the same time looking for the album_id being greater or equal to 10. As the album_id cannot be a value that simultaneously meets that criteria, no rows would be returned. More importantly, this would not be a good fit for the B-tree index. Even if we use the OR operator, it will not be as efficient as having the overlapping range:

```
SELECT *
FROM track
WHERE album_id <= 5 OR album_id >=10;
```

---

3. When To Use

The B-tree index is great for instances when you have values that only repeat a few times, or are completely

unique. Take the track name, for example. There may be a few repeated track names, but for the most part, the names on the tracks are different compared to the total number of rows in the table. As such, a B-tree index would be the best choice. Note that when you are adding indexes to tables, you generally do not have to worry about what the best type of index to use is, because the database will handle it for you. The B-tree index is the default choice.

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

The B-Tree index is the most common type of index, which is used when considering a range of sortable values.

Source: Authored by Vincent Tran